



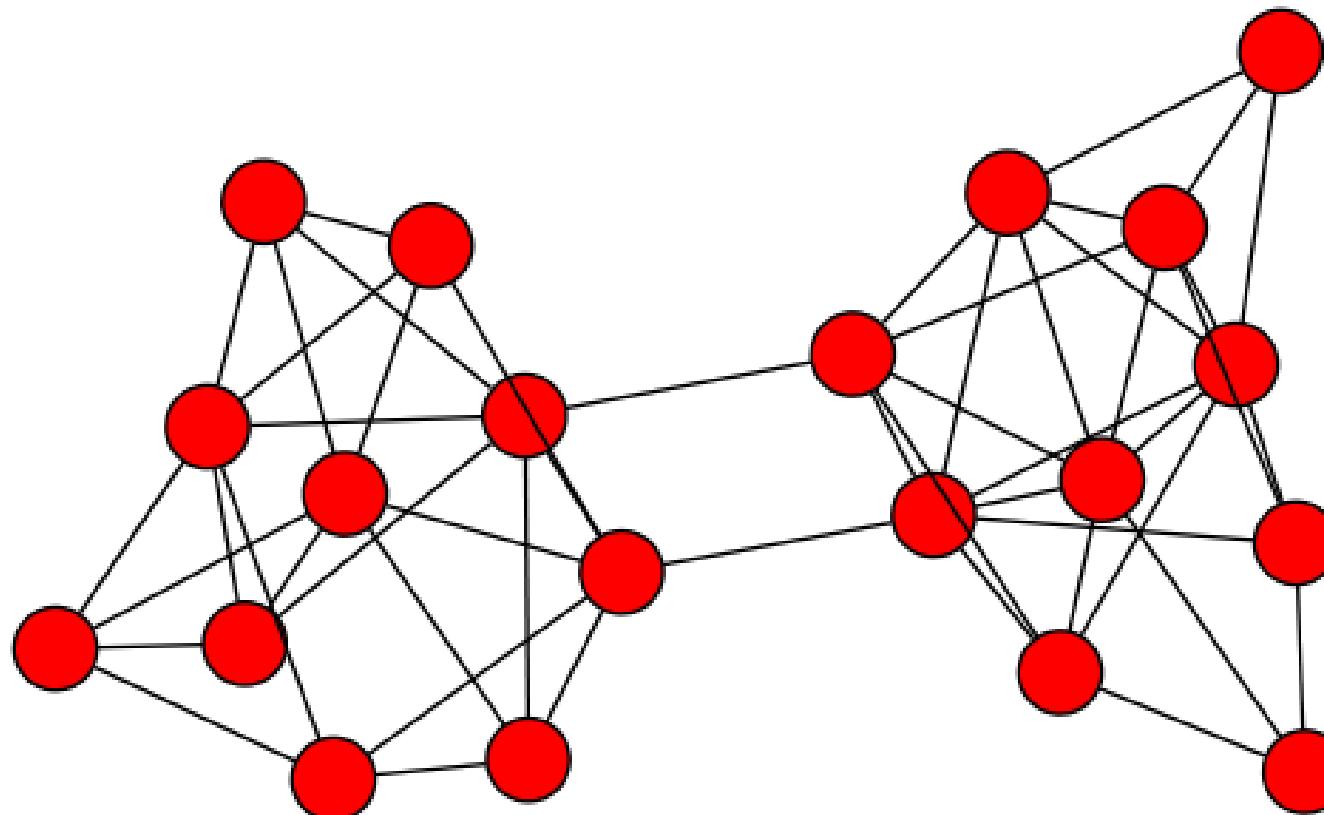
# Frontiers in GNN and Network Embedding

---

Peng Cui, Ziwei Zhang  
Tsinghua University

# Network (Graph)

The general description of data and their relations.

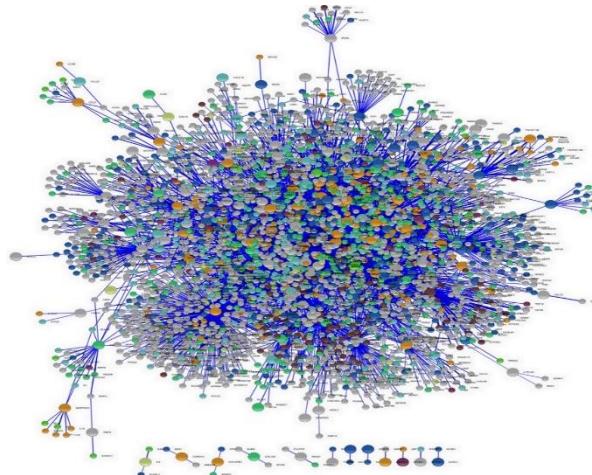


# Many types of data are networks

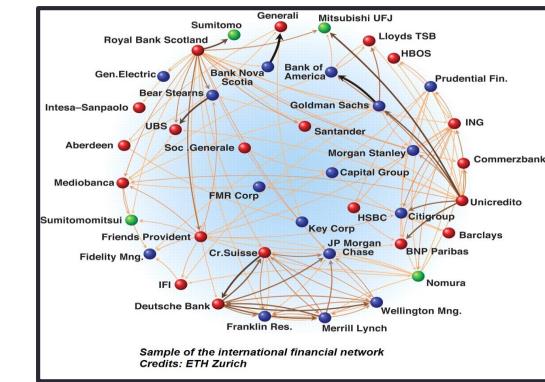
## Social Networks



## Biology Networks



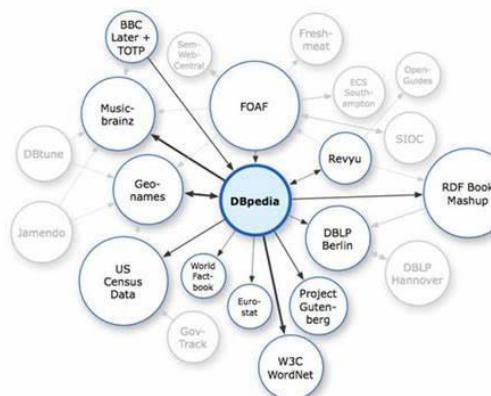
## Finance Networks



## Internet of Things



## Information Networks

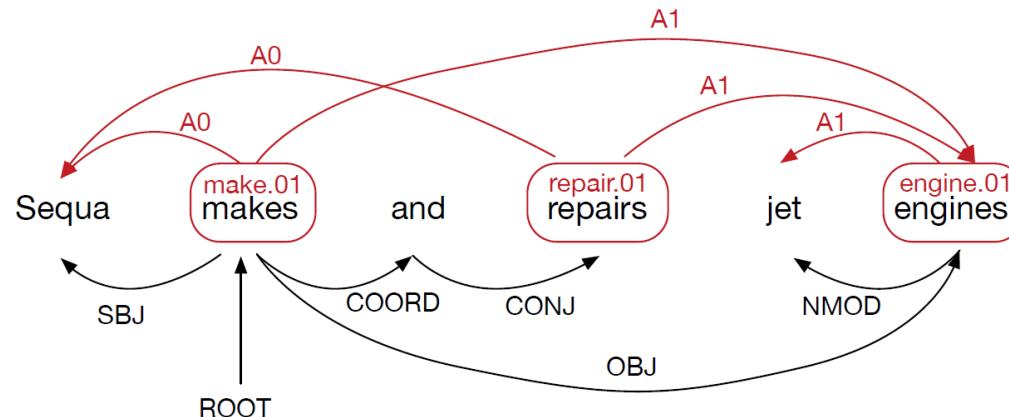


## Logistic Networks

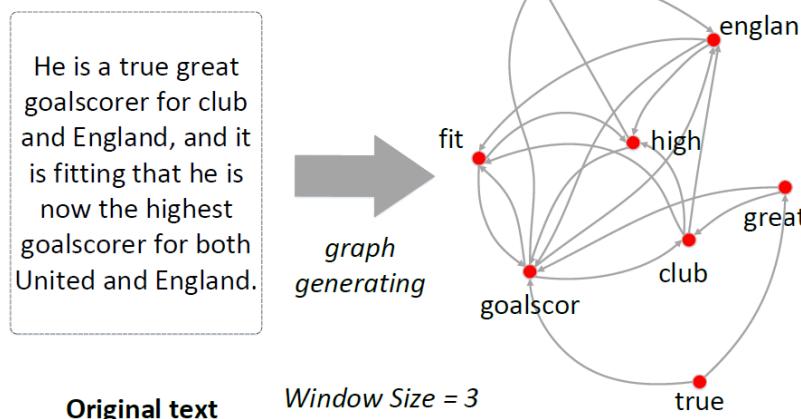


# Graphs in NLP

## Syntactic Dependency



## Word Co-occurrences



## Abstract Meaning Representation

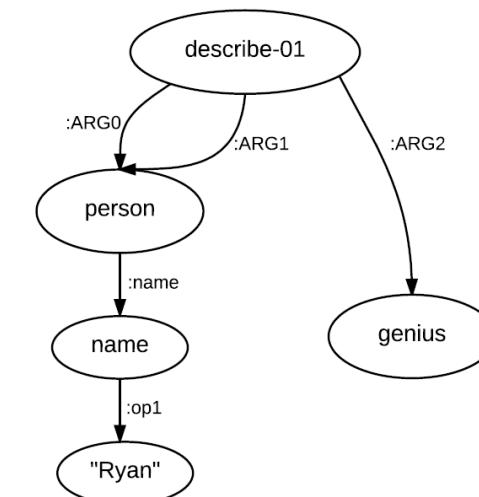
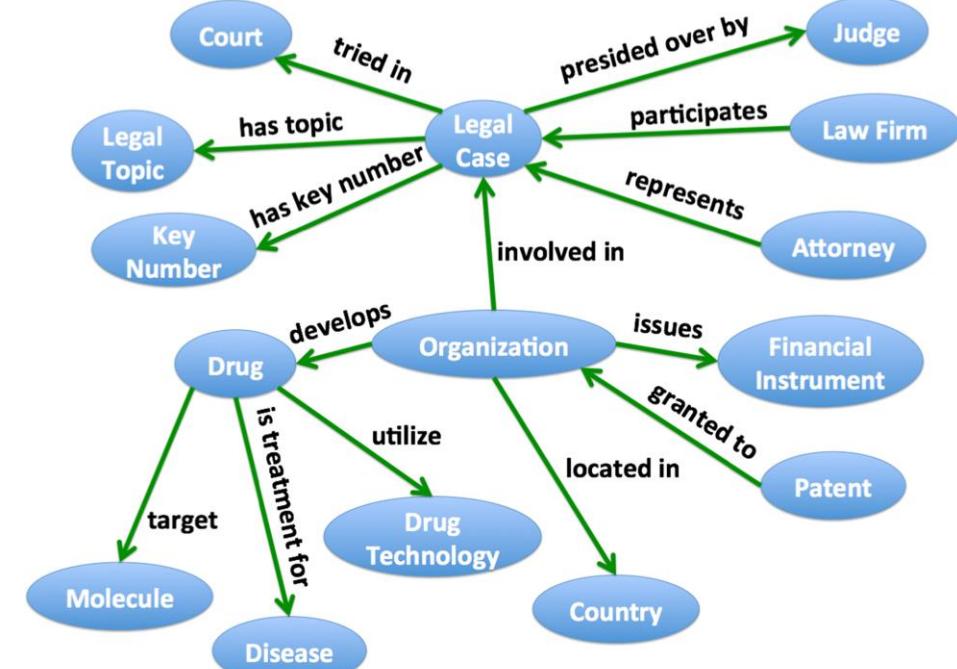
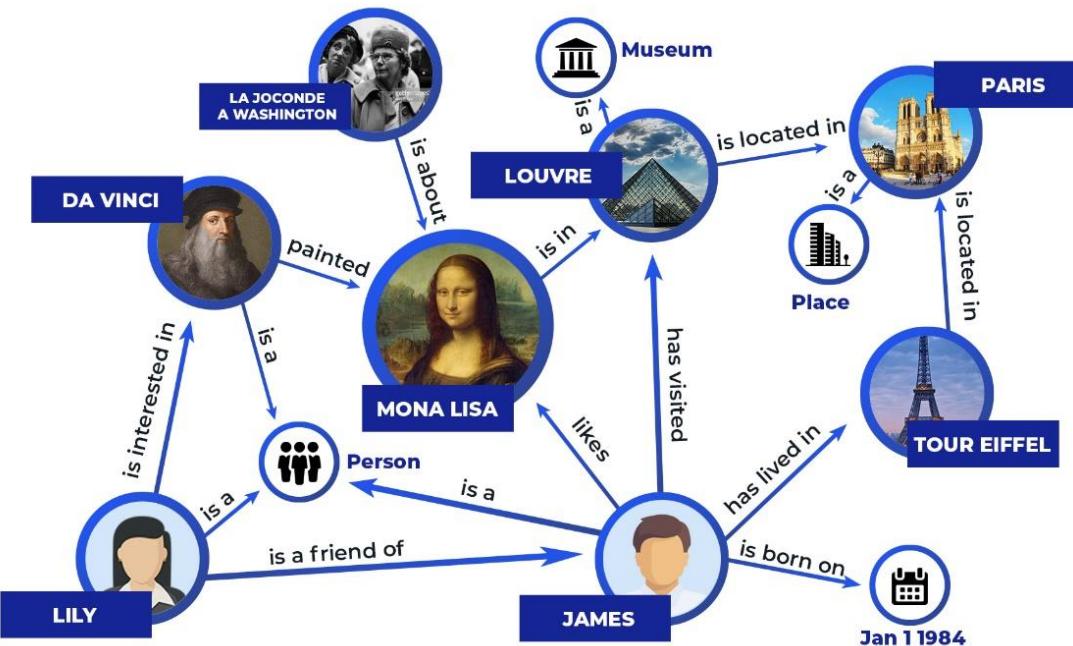


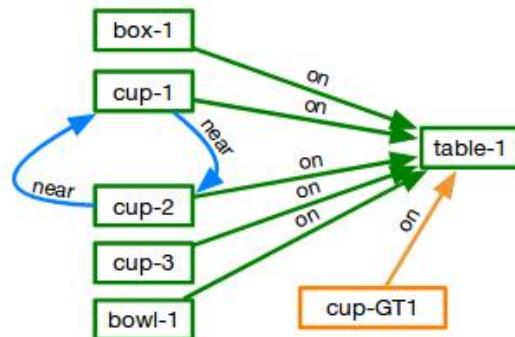
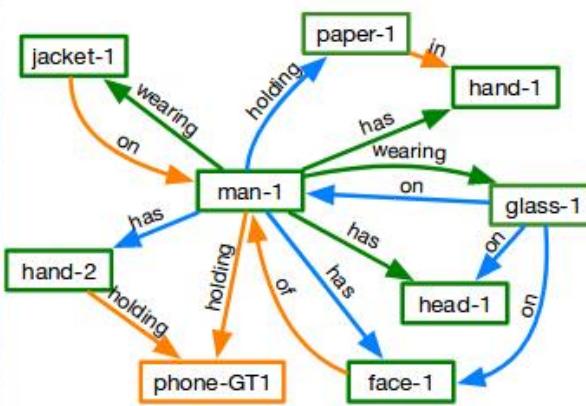
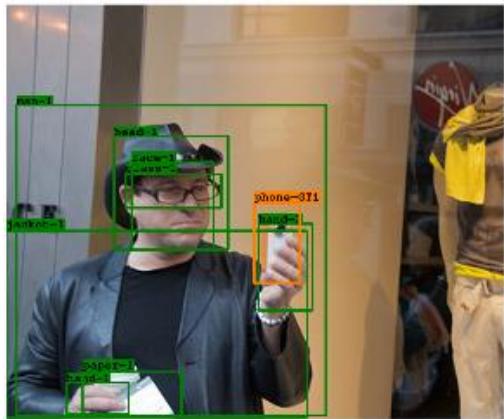
Figure 1: An example of AMR graph meaning “Ryan’s description of himself: a genius.”

# Knowledge Graph



# NLP + Computer Vision

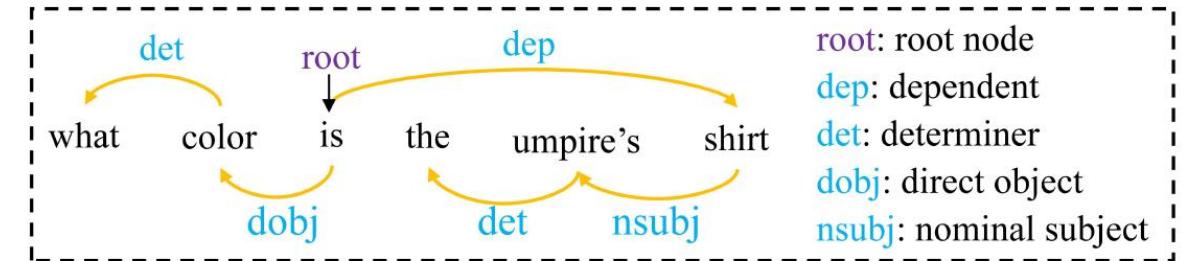
## Scene Graph



## Visual Question Answering



(a) Q: What color is the umpire's shirt (b) Q: What color is the umpire's shirt  
**Ground True Answer: blue**      **Predicted Answer: black**

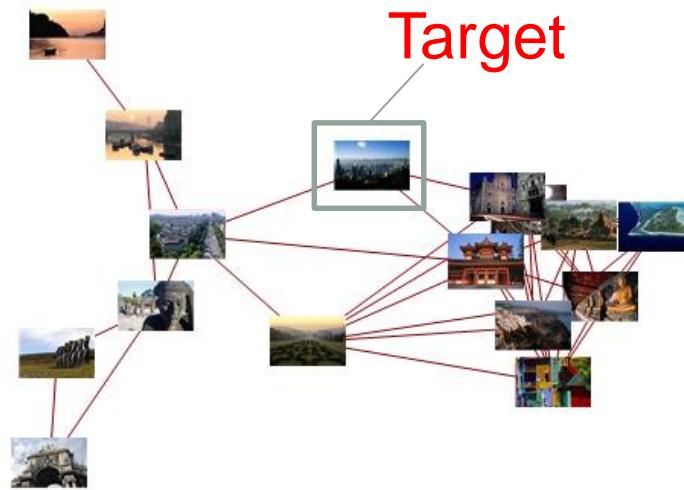


(c) Dependency parsing of the question

# Why network is important?

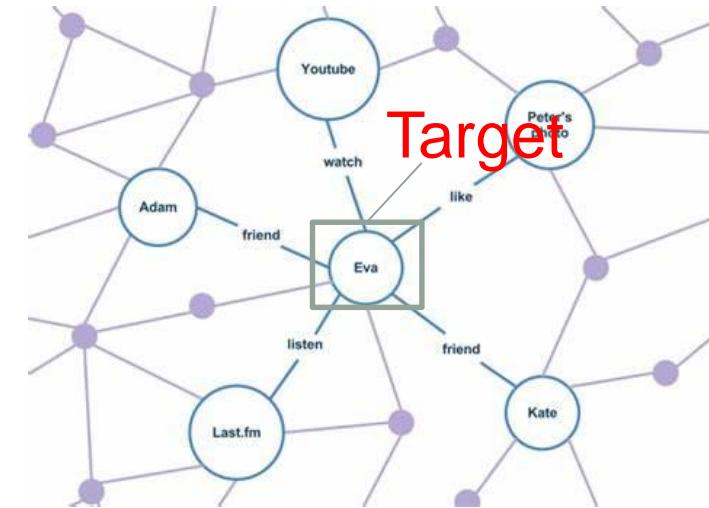
In few cases, you only care about a subject but not its relations with other subjects.

**Image Characterization**



Reflected by relational subjects

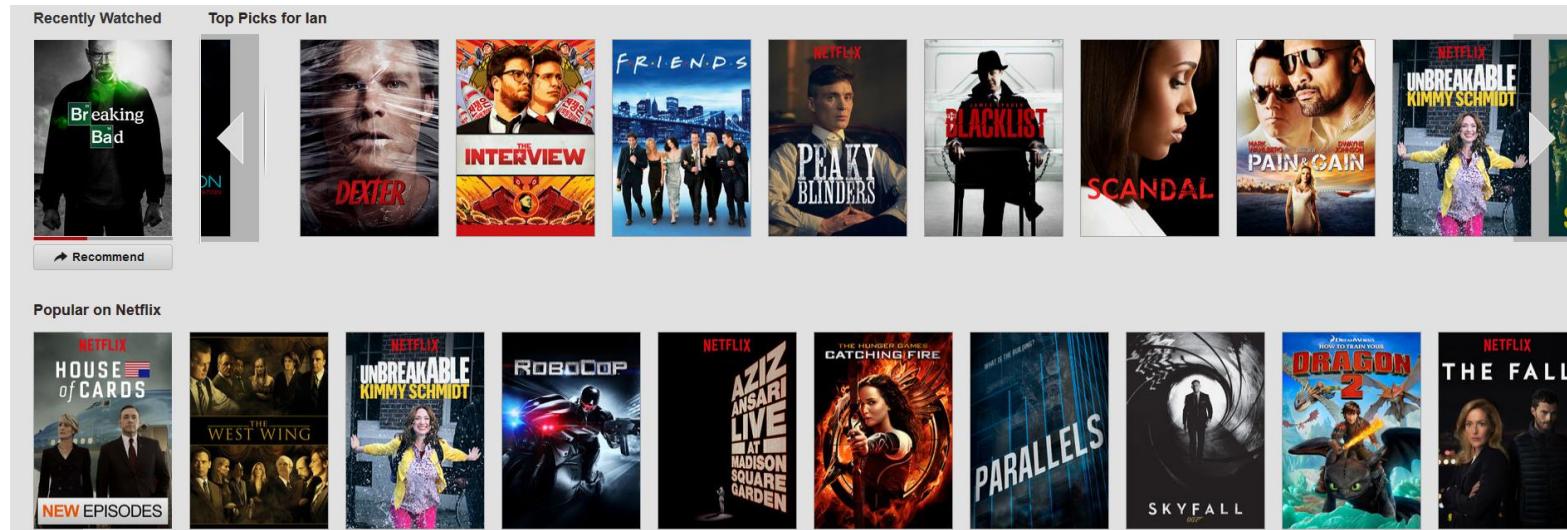
**Social Capital**



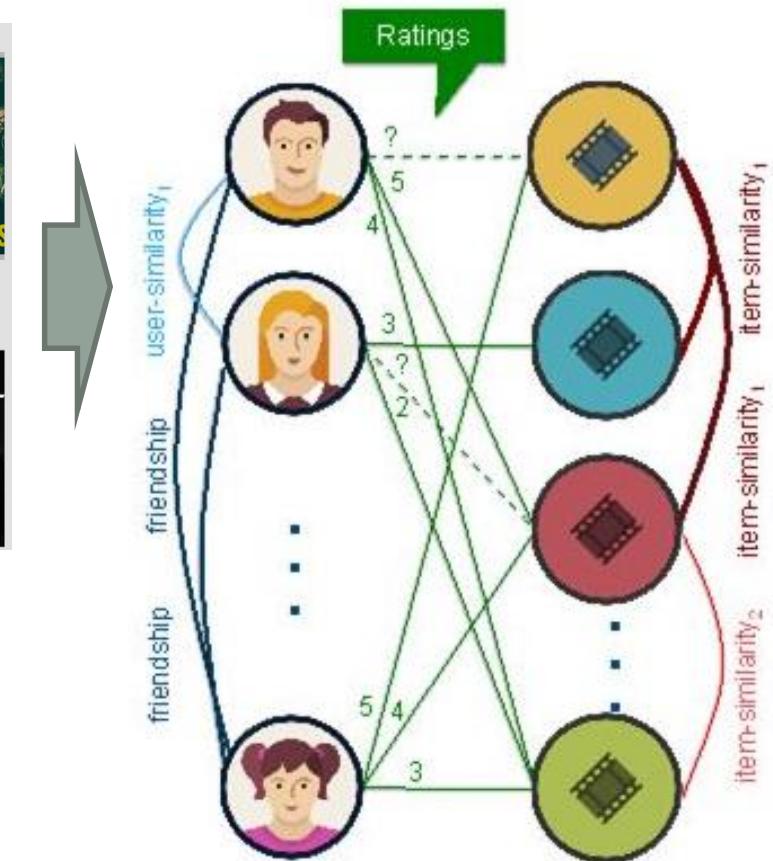
Decided by relational subjects

# Many applications are intrinsically network problems

## Recommendation Systems



## Link prediction in bipartite graphs

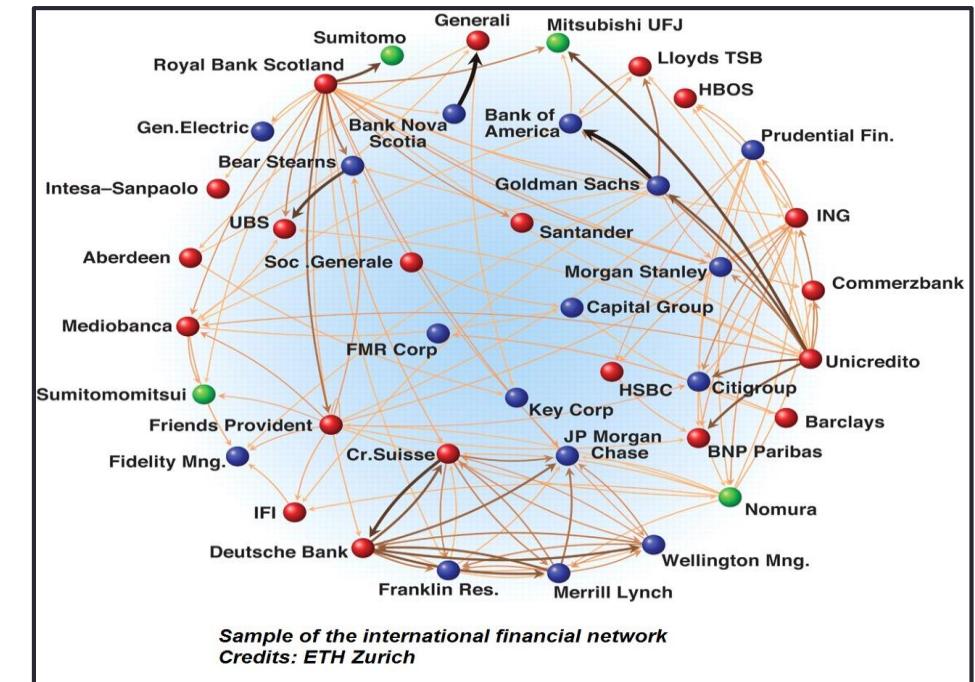


# Many applications are intrinsically network problems

## Financial credit & risk management



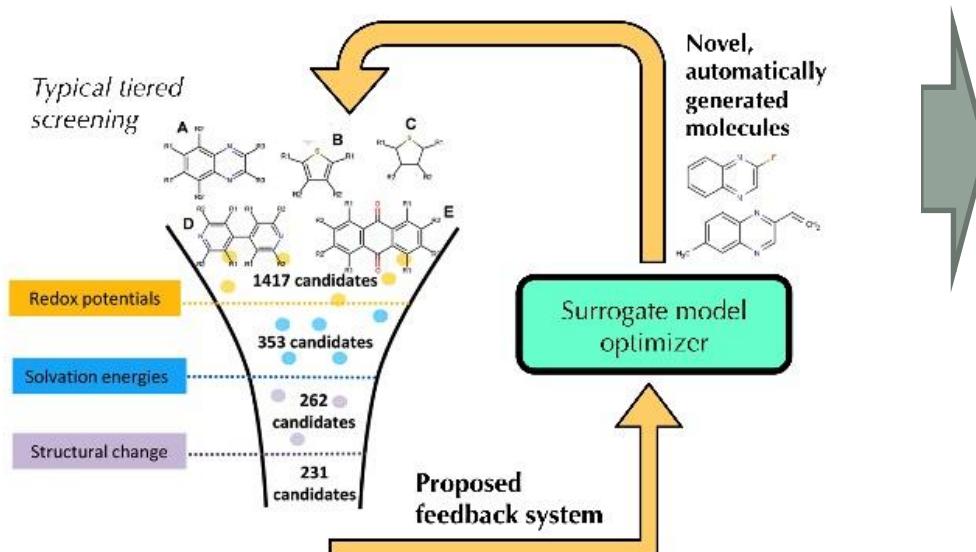
## Node importance & classification



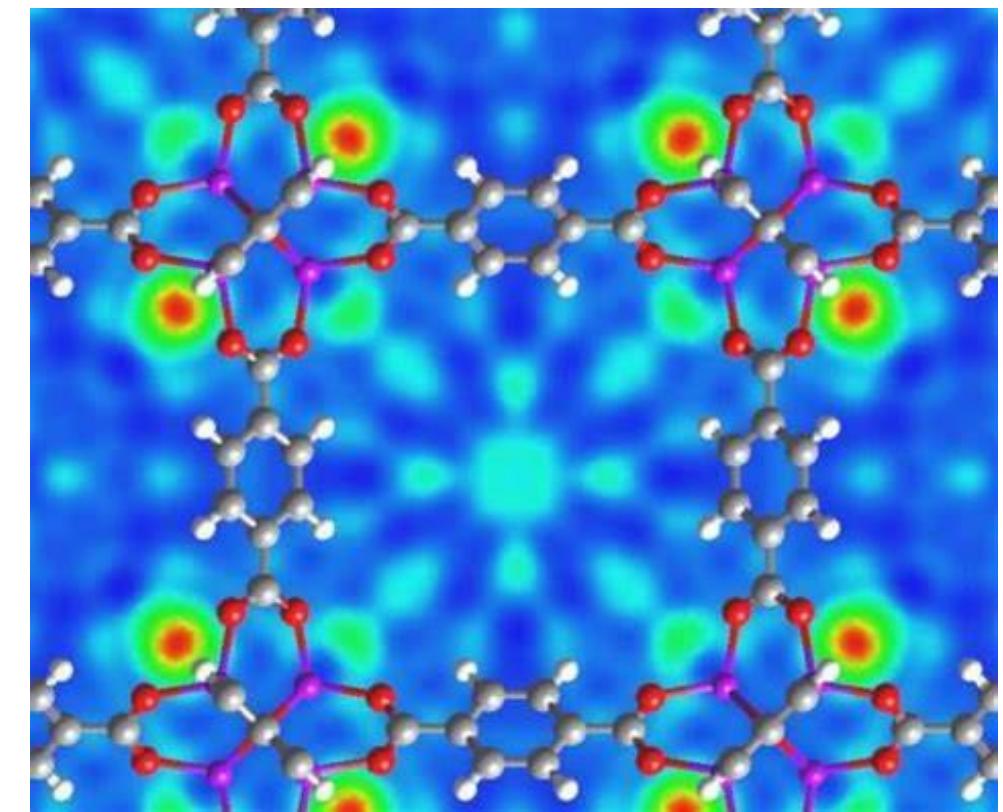
# Many applications are intrinsically network problems

## New material discovery

### Materials discovery engine concept



## Subgraph pattern discovery

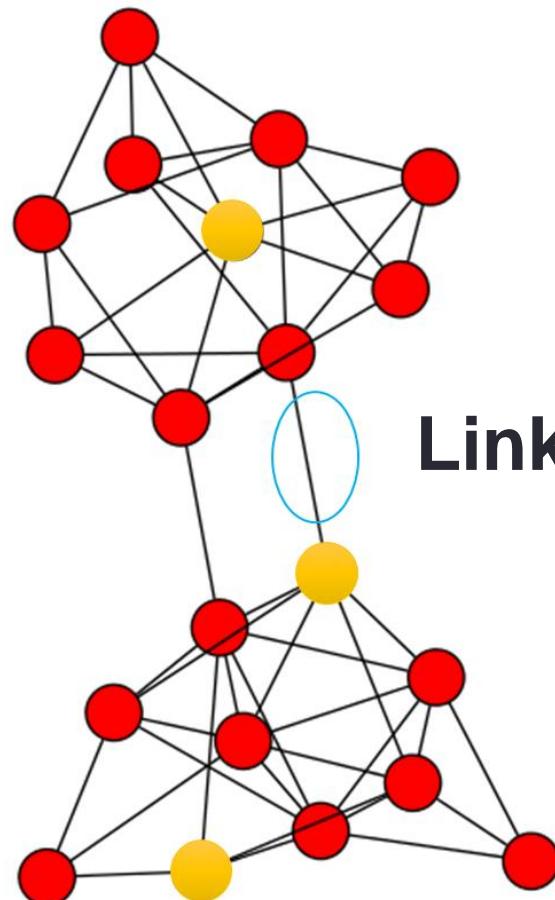


# Graph as a data model

- The last resort for the curse of complexity in real applications
  - Geographical networks, relationships, etc.
- Divide-and-conquer in modeling
  - Individual nodes and edges are well structured
  - Global structures are weakly organized

# Networks are not *learning-friendly*

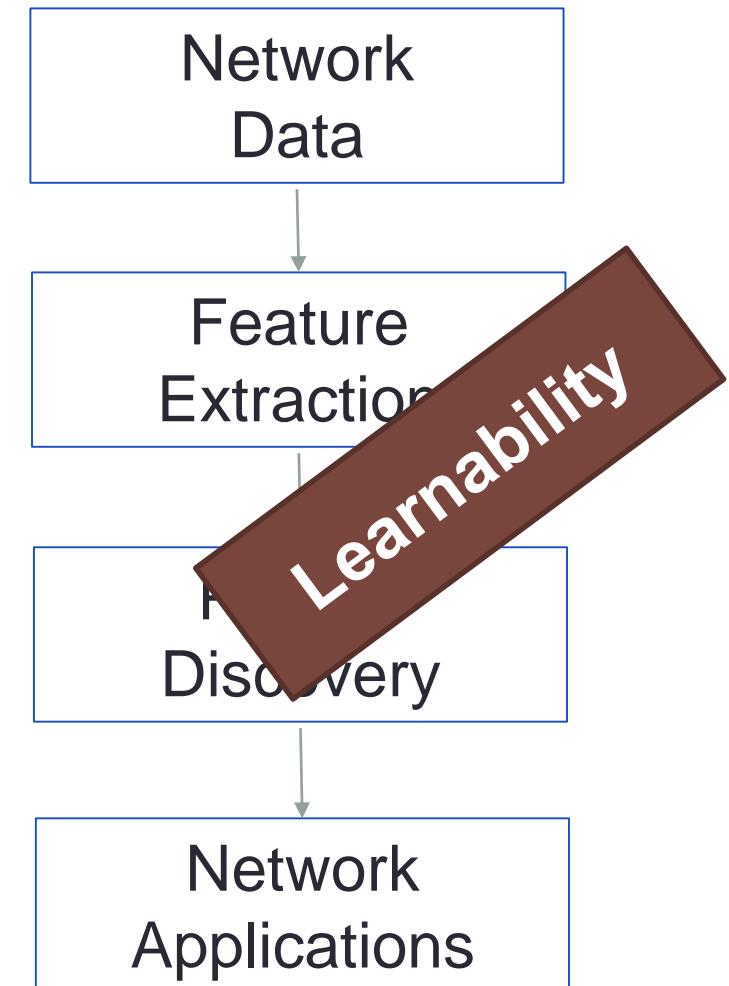
$$G = (V, E)$$



Inapplicability of  
ML methods

Links → Topology

Pipeline for network analysis

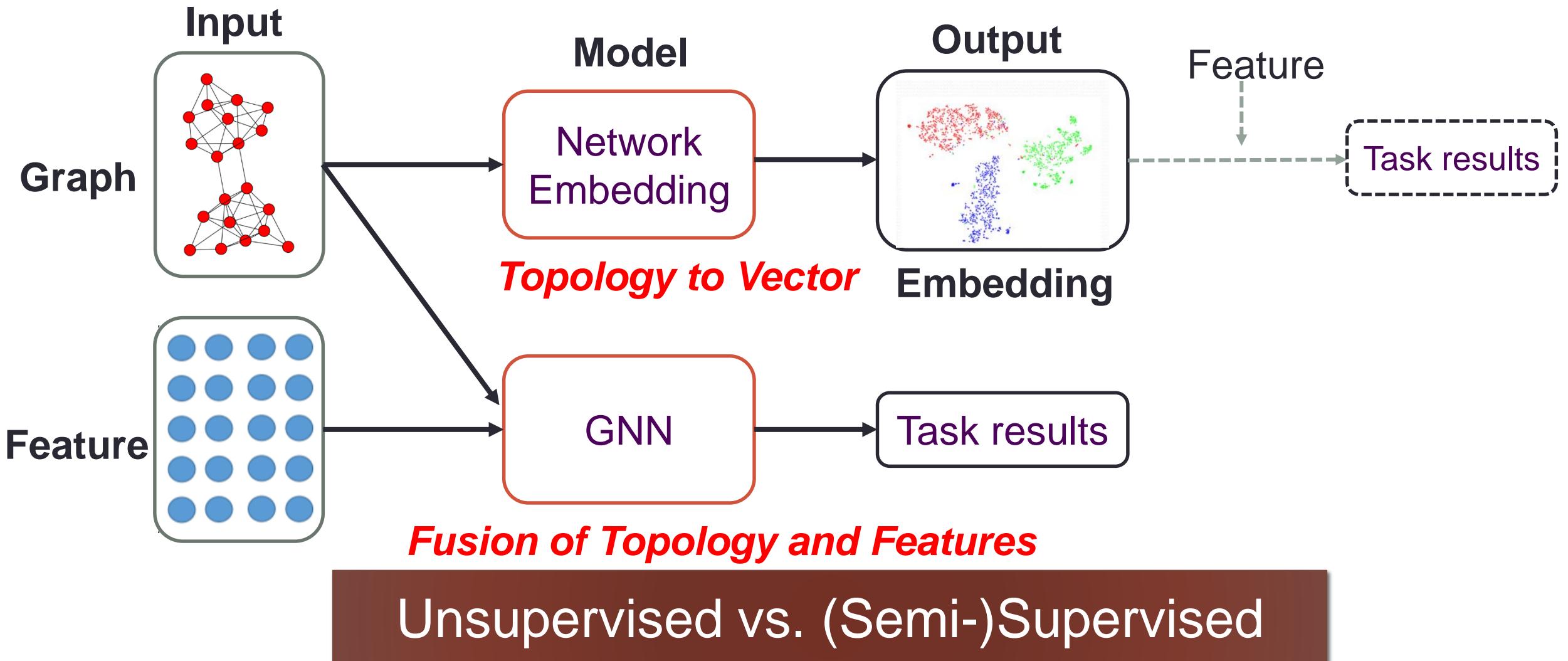


# Learning from networks

Network  
Embedding

GNN

# Network Embedding and GNN

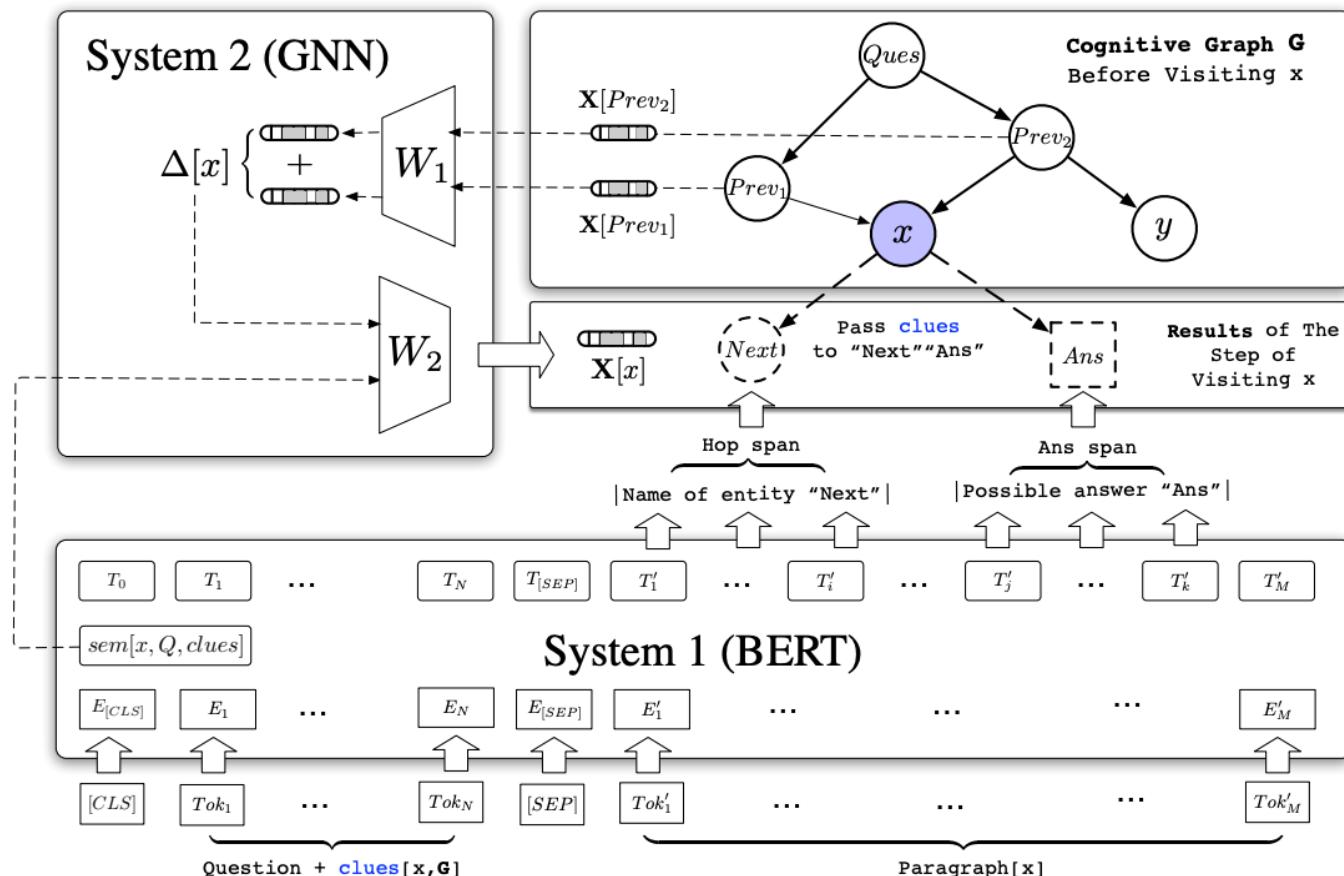


# Graph Neural network vs. Network embedding

- In some sense, they are different
- **Graphs** exist in *mathematics* (Data Structure)
  - Mathematical structures used to model pairwise relations between objects
- **Networks** exist in the *real world* (Data)
  - Social networks, logistic networks, biology networks, transaction networks, etc.
- A network can be represented by a graph
- A dataset that is not a network can also be represented by a graph

# GNN for Natural Language Processing

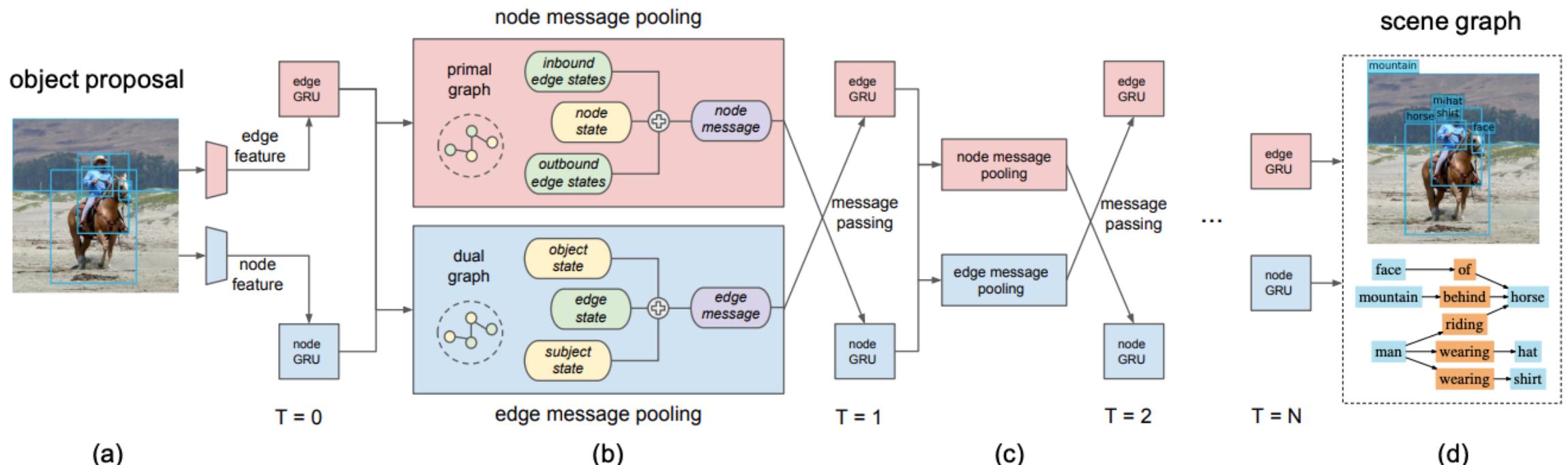
- Many papers on BERT + GNN.
- BERT is for retrieval.
  - It creates an initial graph of relevant entities and the initial evidence.
- GNN is for reasoning.
  - It collects evidence (i.e., old messages on the entities) and arrive at new conclusions (i.e., new messages on the entities), by passing the messages around and aggregating them.



Cognitive Graph for Multi-Hop Reading Comprehension at Scale. Ding et al., ACL 2019.  
 Dynamically Fused Graph Network for Multi-hop Reasoning. Xiao et al., ACL 2019.

# GNN for Computer Vision

- A popular trend in CV is to construct a graph during the learning process
  - To process multiple objects or parts in a scene, and to infer their relationships
- Example: Scene graphs



Scene Graph Generation by Iterative Message Passing. Xu et al., CVPR 2017.  
 Image Generation from Scene Graphs. Johnson et al., CVPR 2018.

# GNN for Symbolic Reasoning

- We can view the process of symbolic reasoning as a directed acyclic graph
- Many recent efforts use GNNs to perform symbolic reasoning

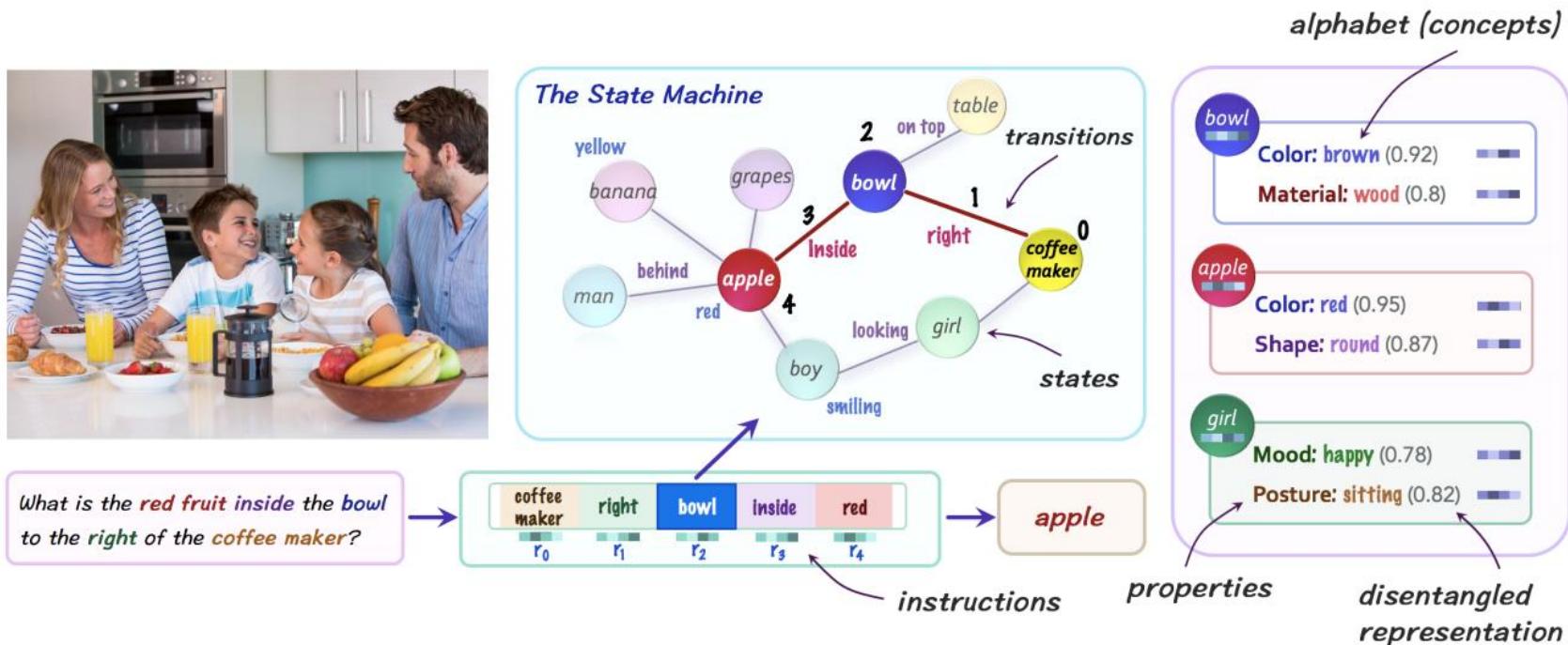


Figure 1: The Neural State Machine is a graph network that simulates the computation of an automaton.

Learning by Abstraction: The Neural State Machine. Hudson & Manning, NeurIPS 2019  
 Can Graph Neural Networks Help Logic Reasoning? Zhang et al., arXiv 1906.02111  
 Symbolic Graph Reasoning Meets Convolutions. Liang et al., NeurIPS 2018

# GNN for Structural Equation Modeling

- Structural equation modeling, a form of causal modeling, tries to describe the relationships between the variables as a directed acyclic graph (DAG)
- GNN can be used to represent a nonlinear structural equation and help find the DAG, after treating the adjacency matrix as parameters

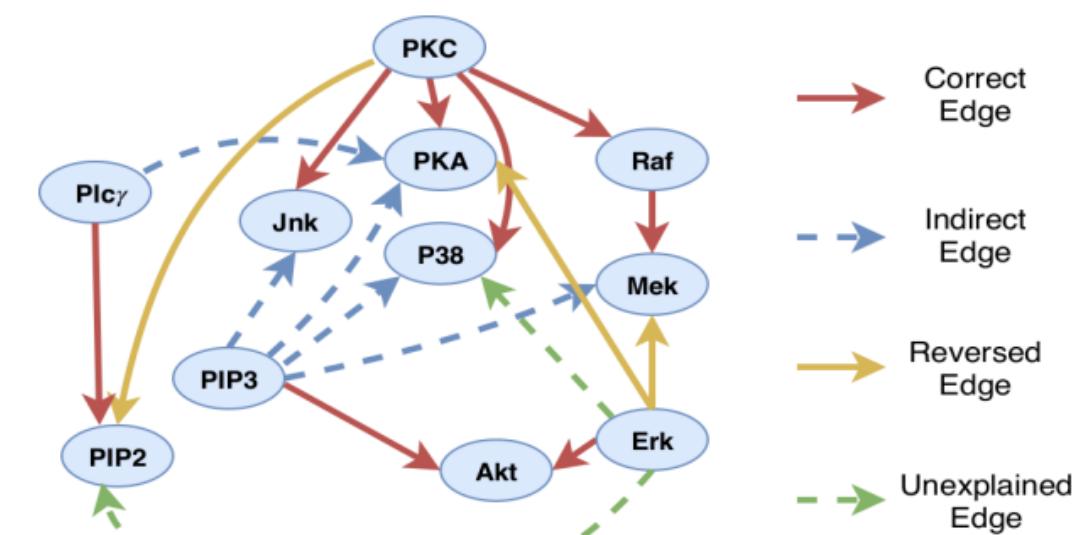
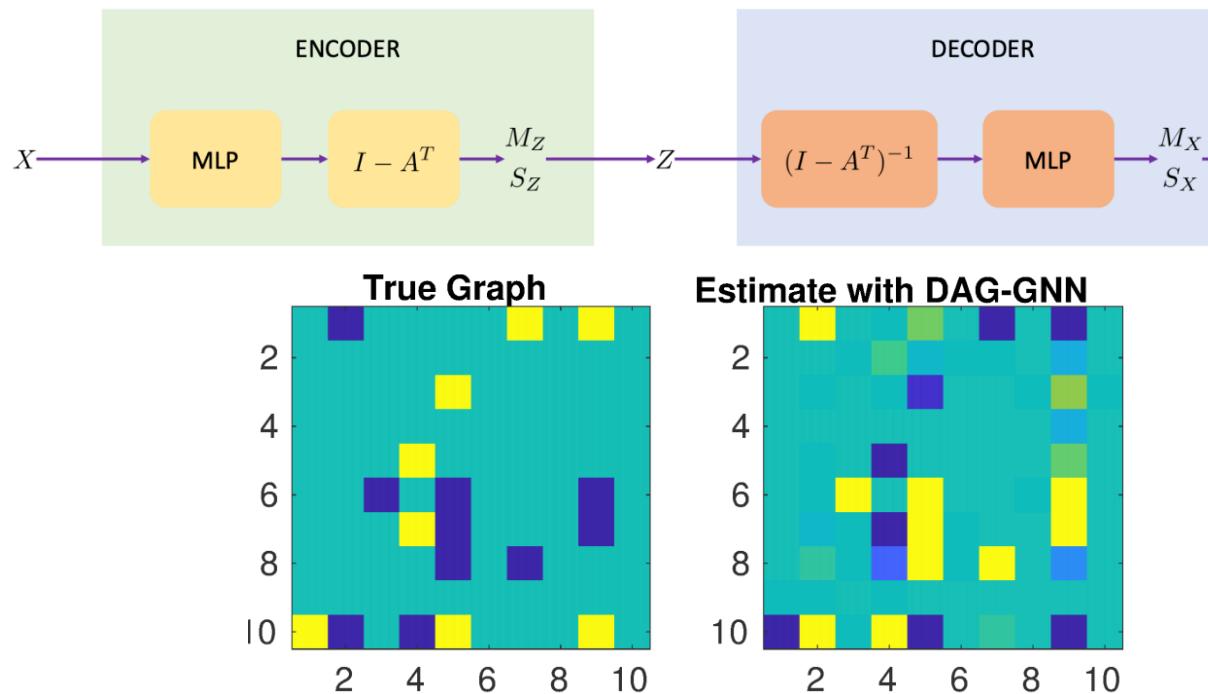
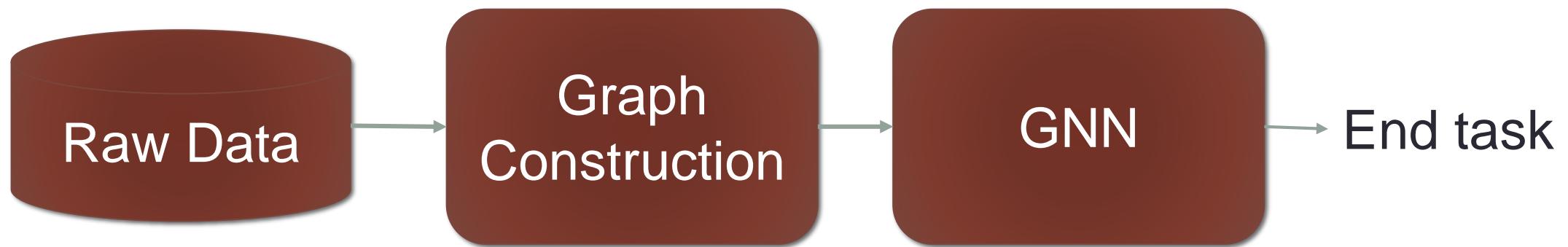


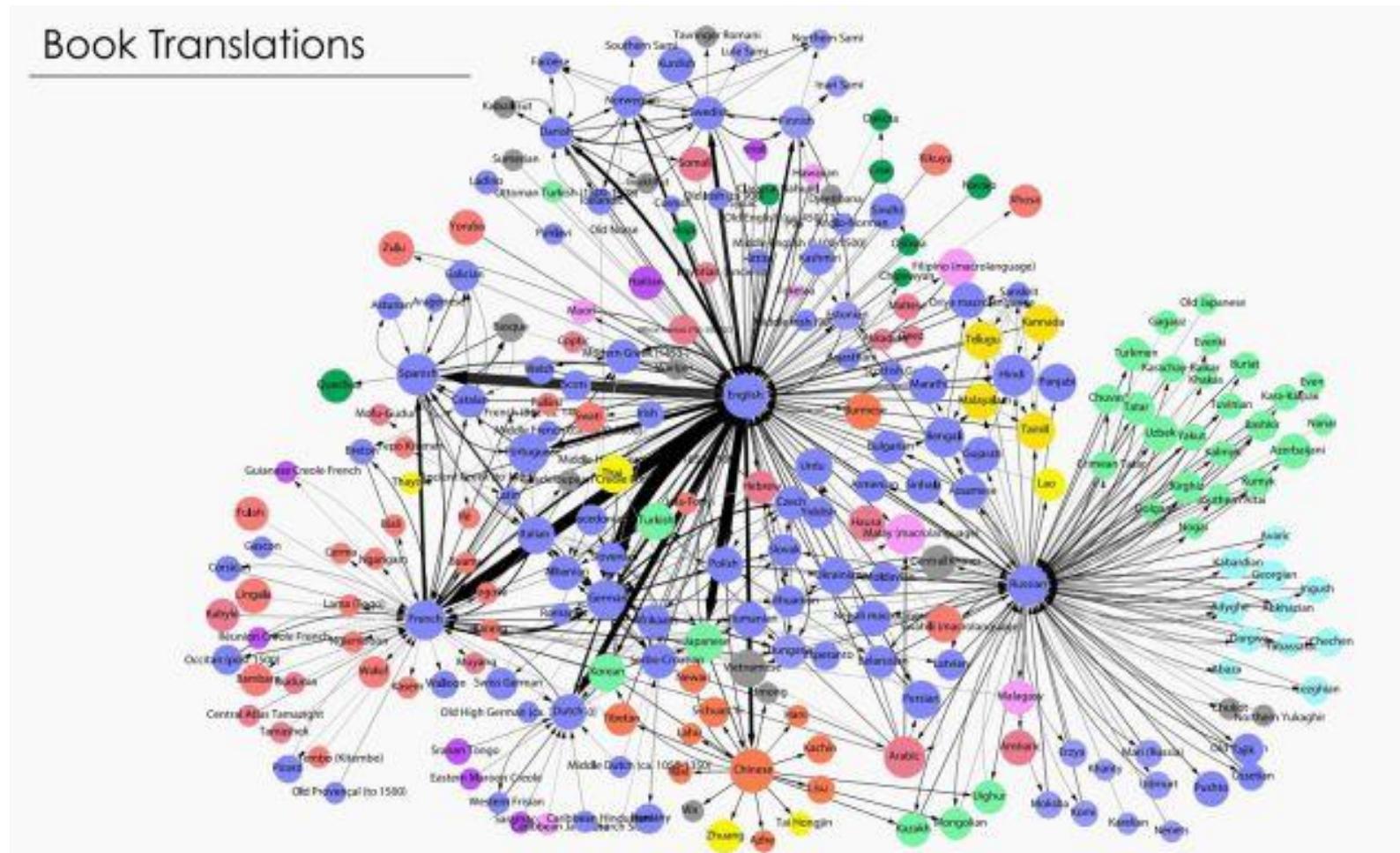
Figure 8. Estimate protein signaling network.

# Pipeline for (most) GNN works



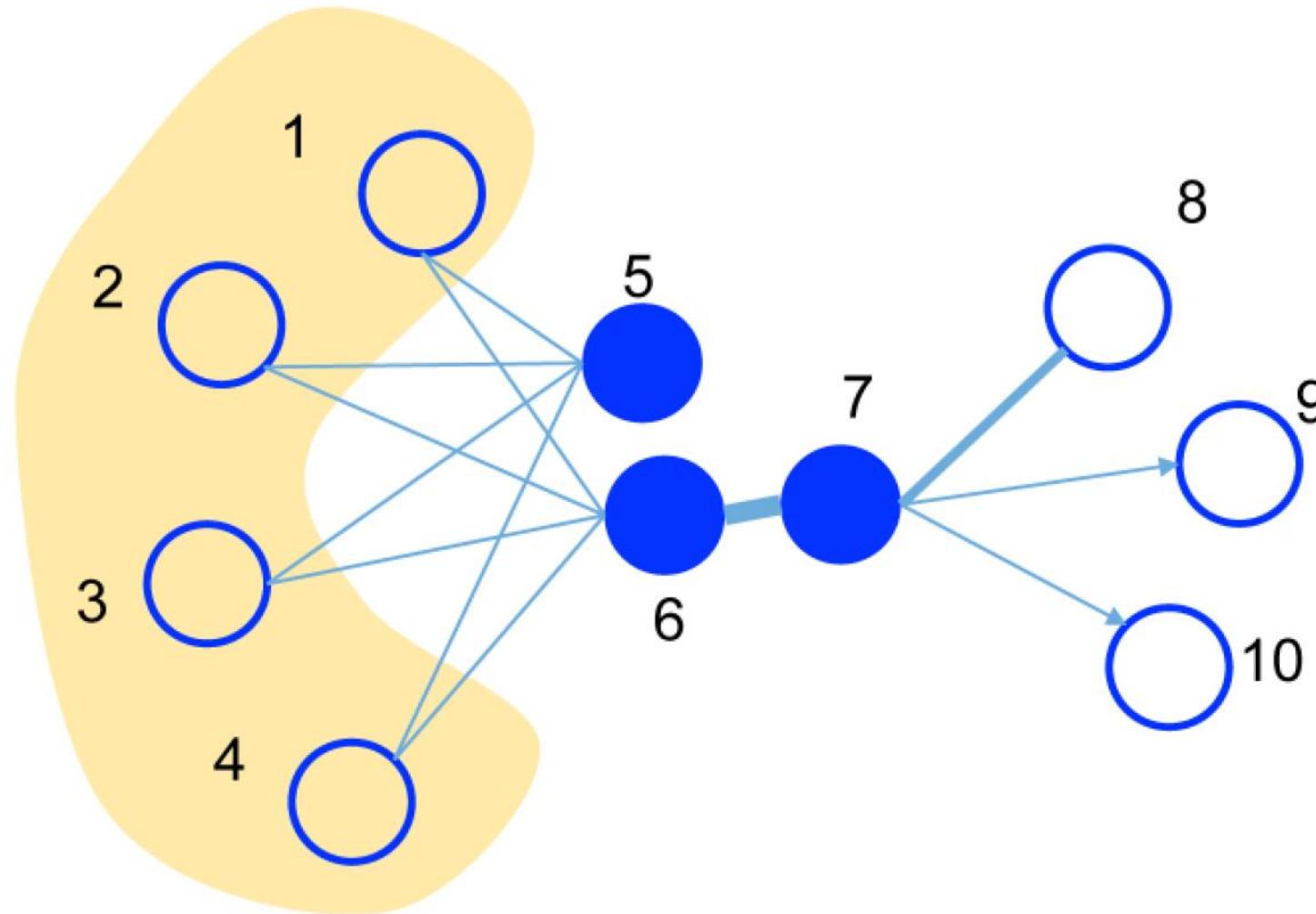
# Network embedding: topology to vector

## □ Co-occurrence (neighborhood)



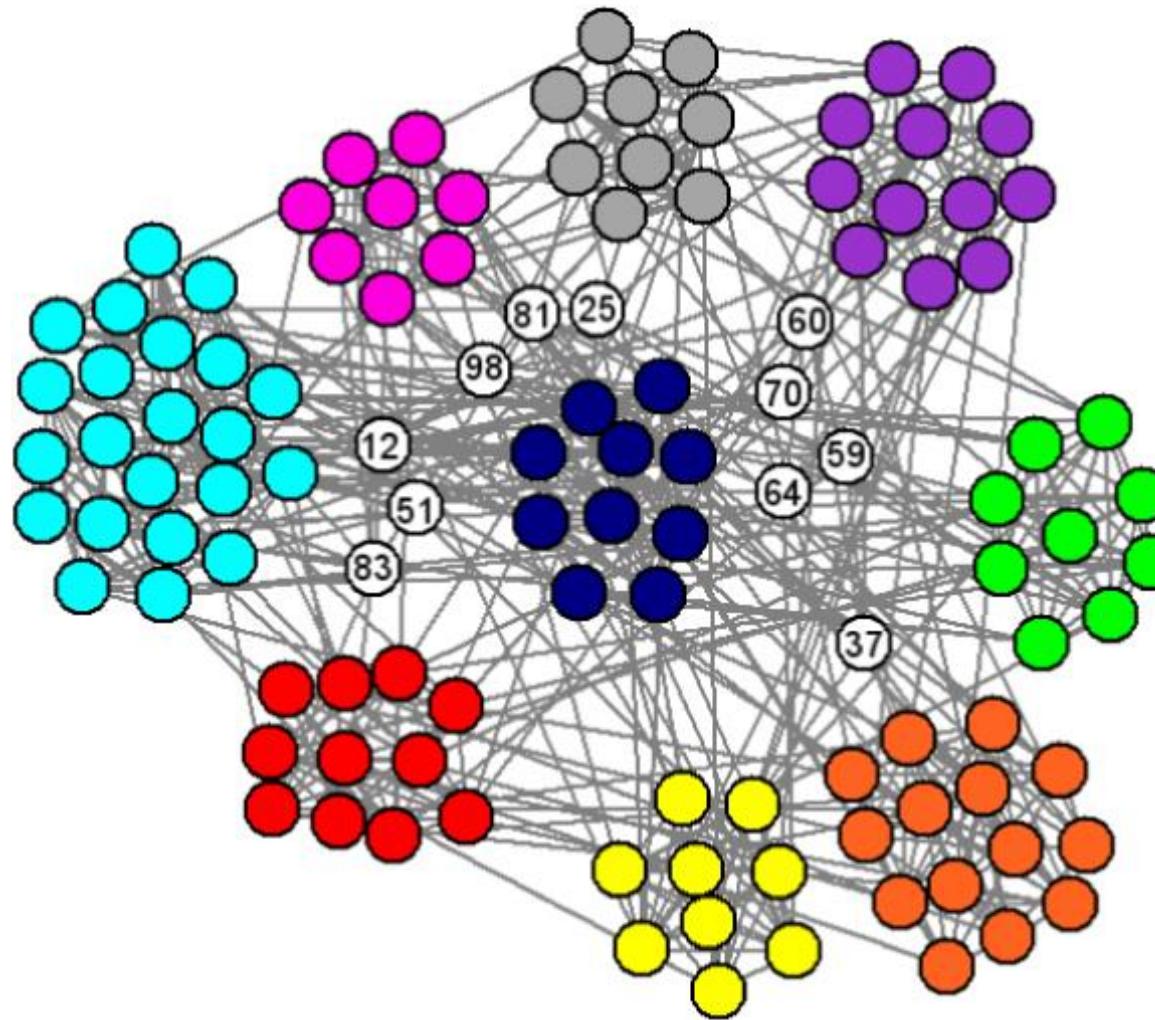
# Network embedding: topology to vector

## □ High-order proximities



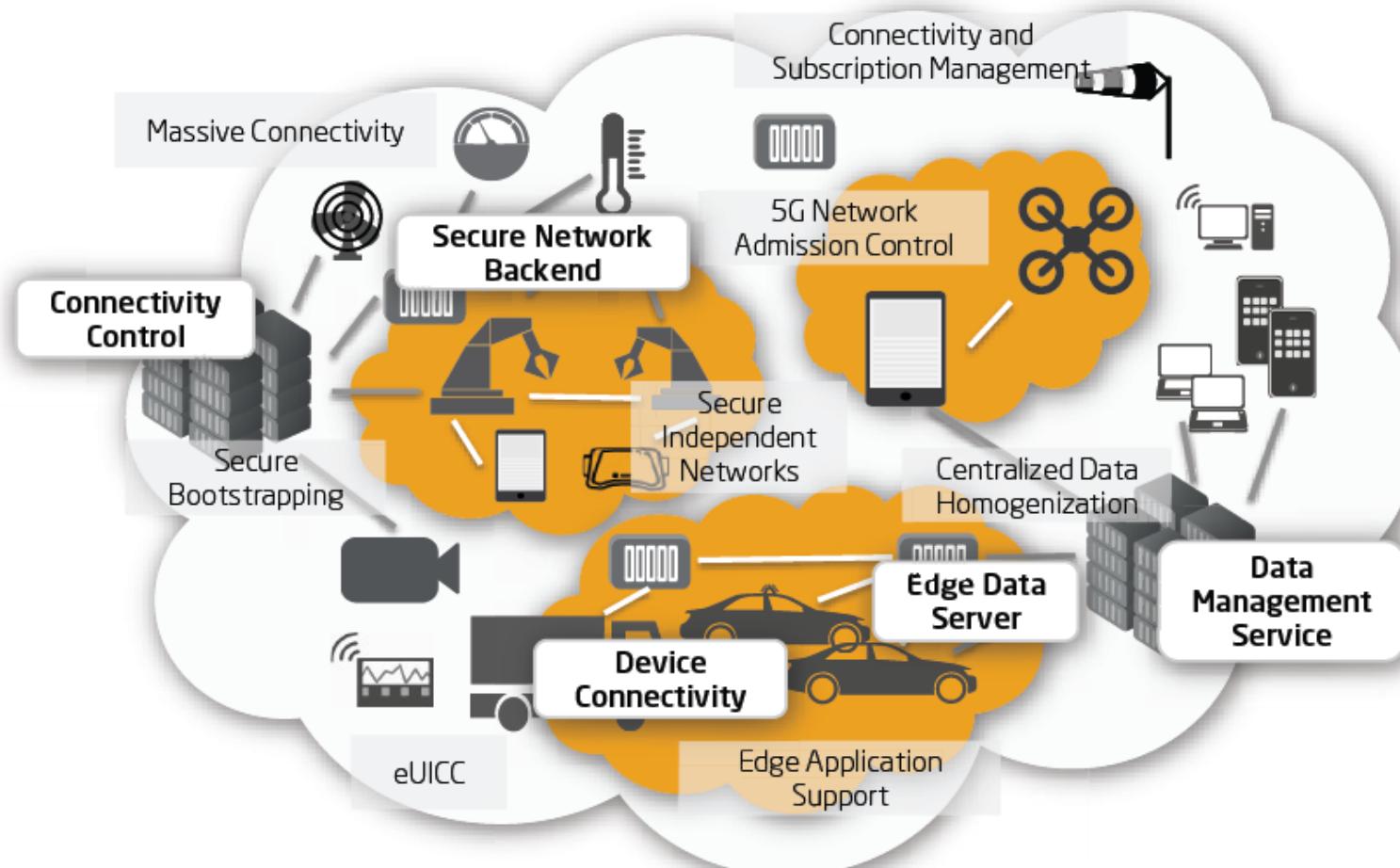
# Network embedding: topology to vector

## □ Communities

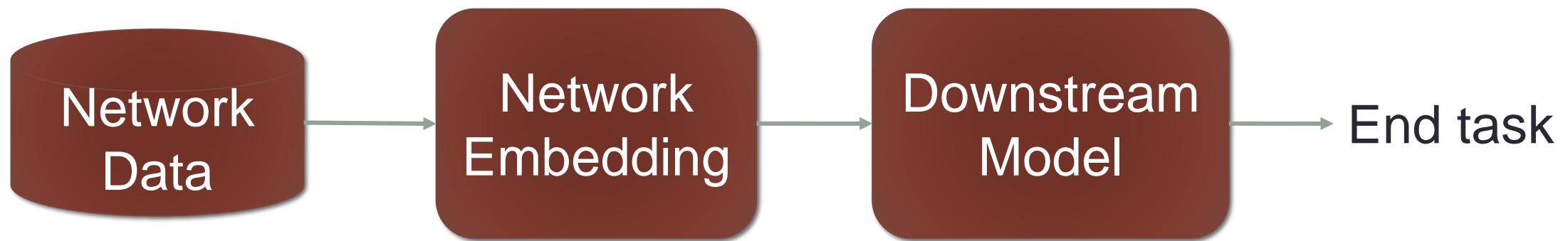


# Network embedding: topology to vector

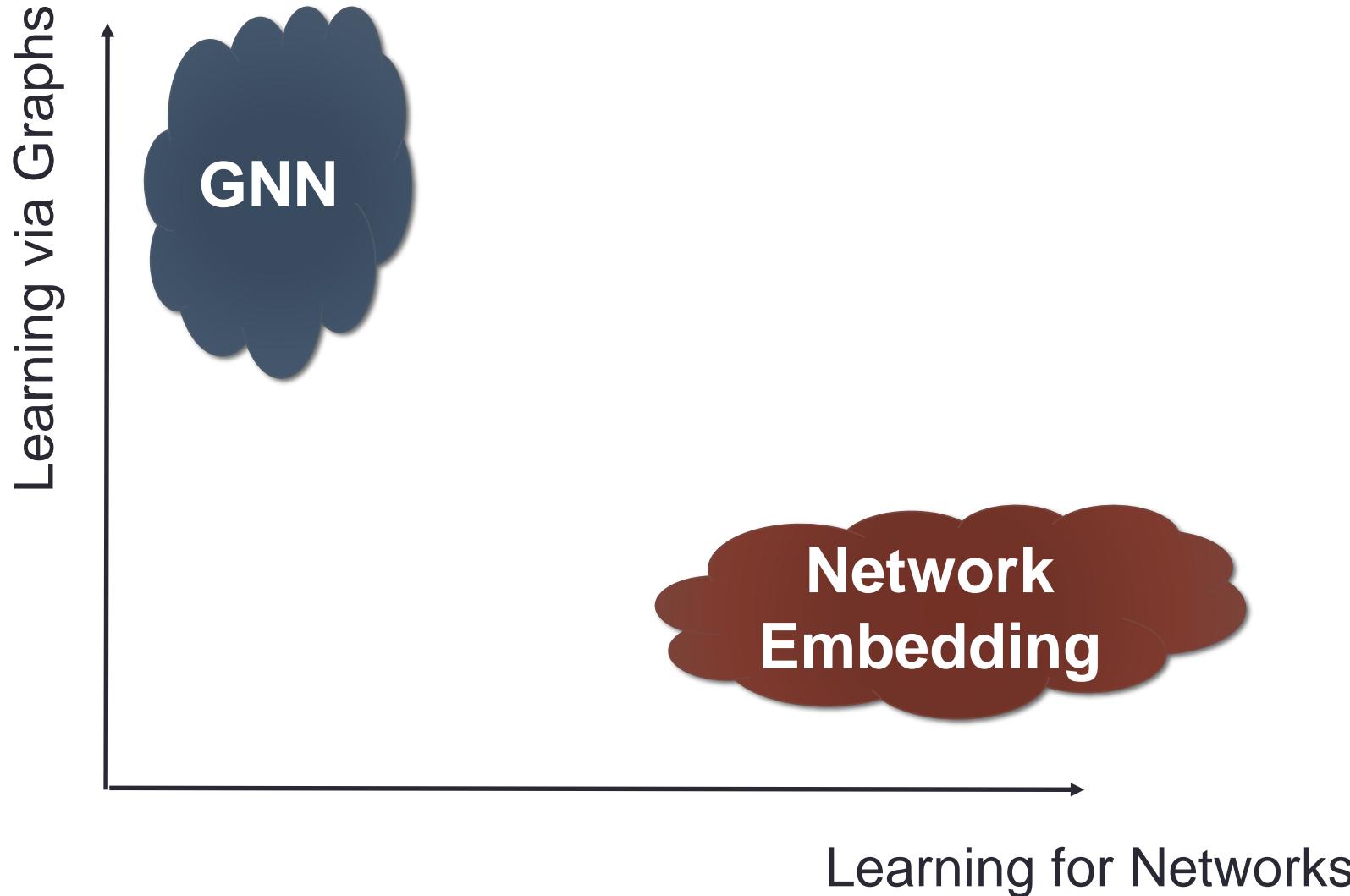
## □ Heterogeneous networks



# Pipeline for (most) Network Embedding works

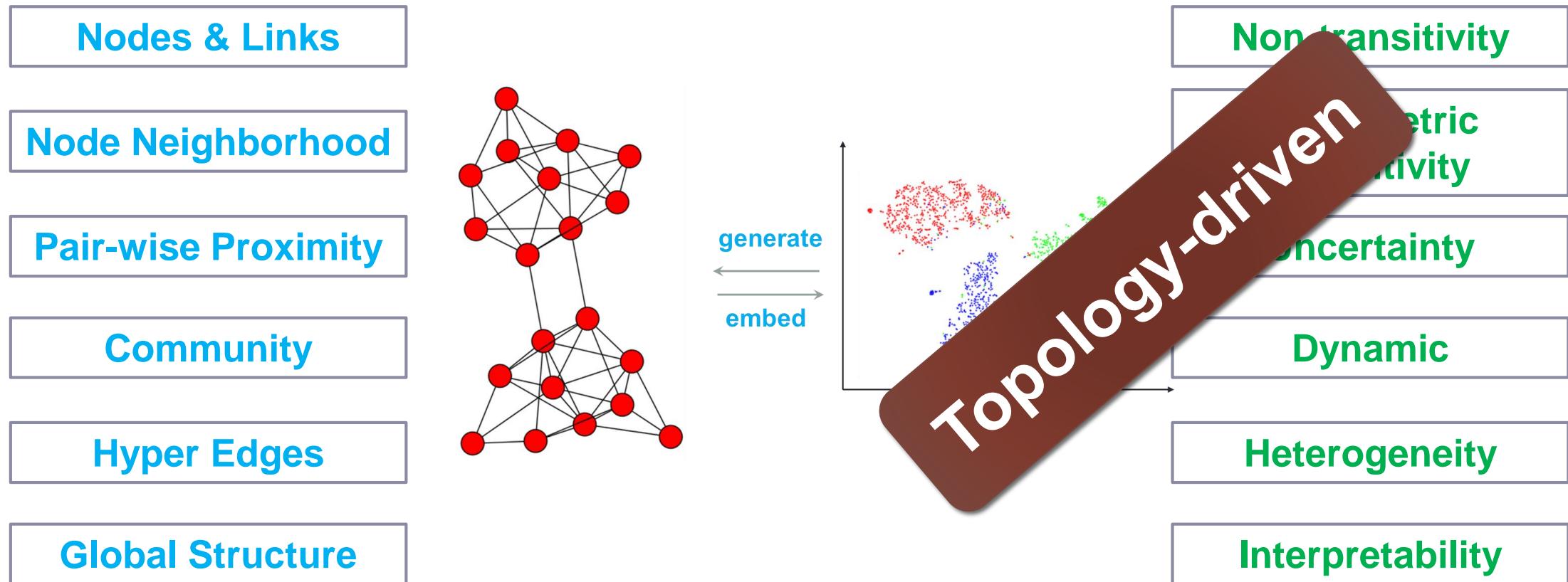


# Learning for Networks vs. Learning via Graphs



# The intrinsic problems NE is solving

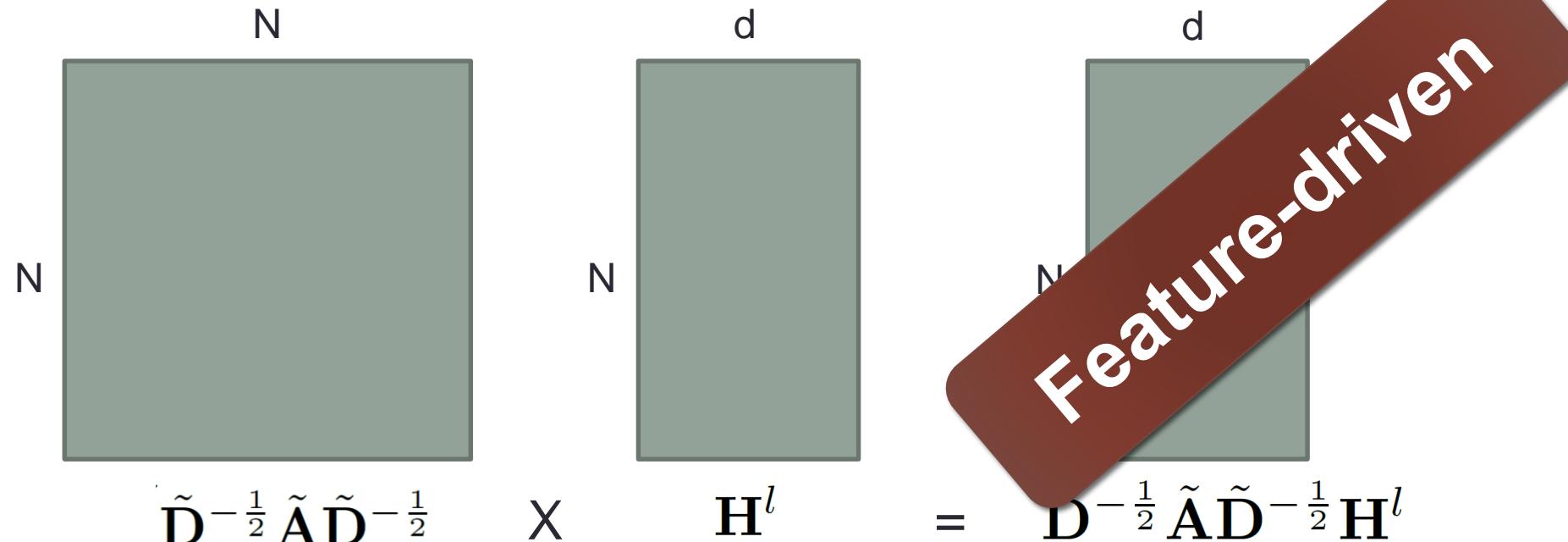
Reducing representation dimensionality while preserving necessary topological **structures** and **properties**.



# The intrinsic problem GNN is solving

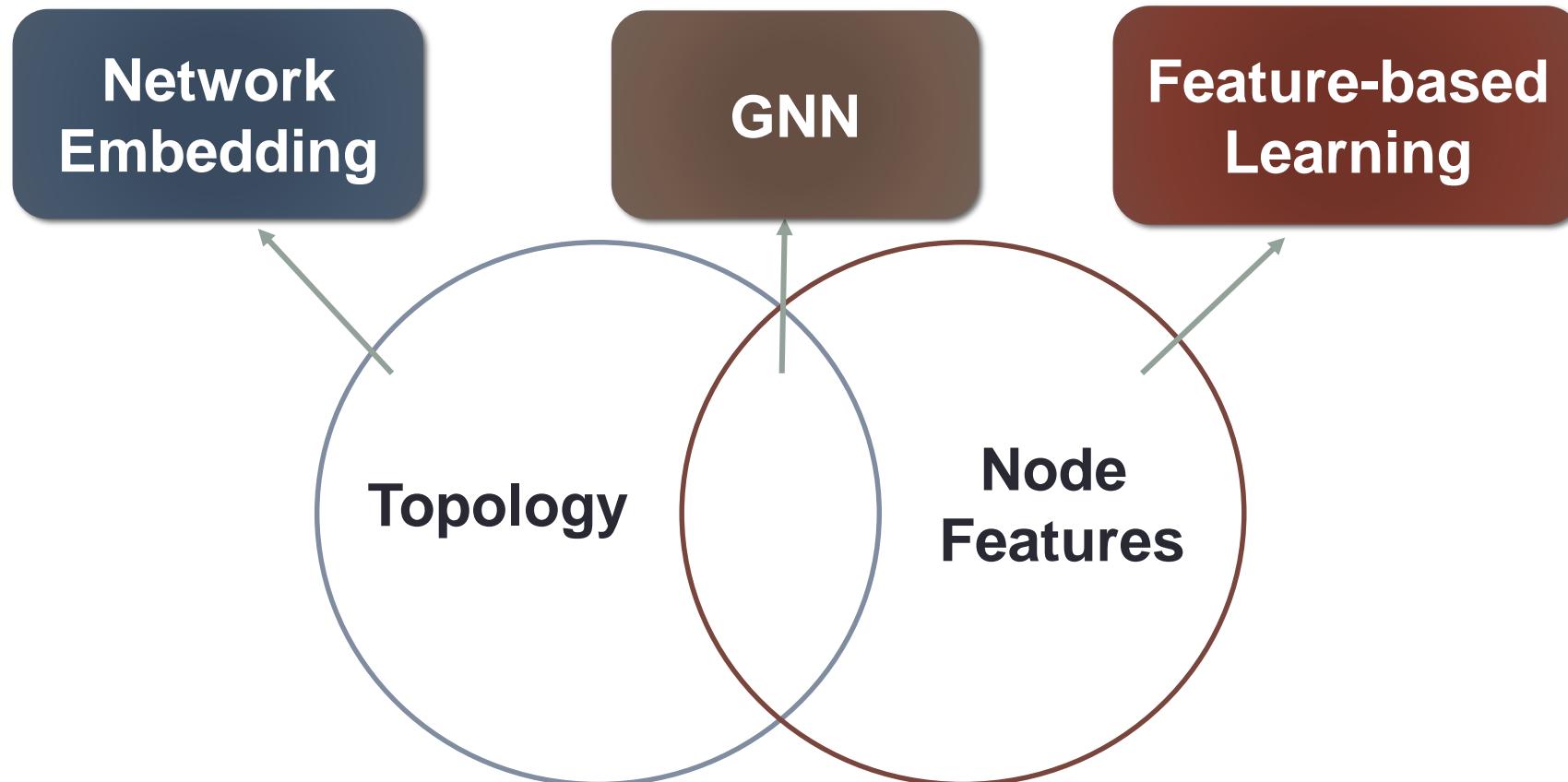
Fusing topology and features in the way of **smoothing features** with the assistance of topology.

$$\mathbf{H}^{l+1} = \rho \left( \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^l \Theta^l \right)$$



# Network Embedding vs. GNN

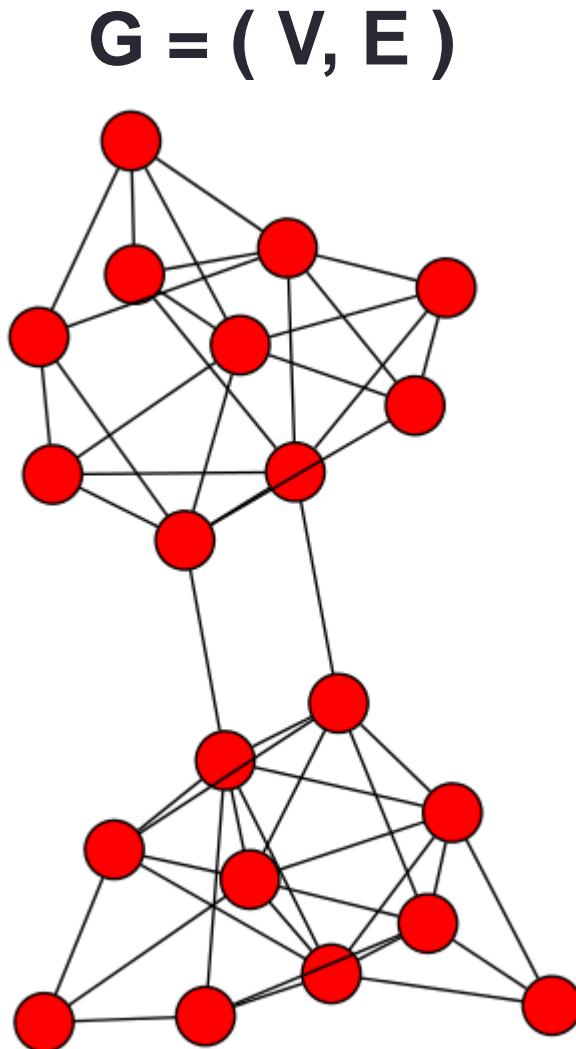
There is no better one, but there is more proper one.



# Learning from networks

Network  
Embedding

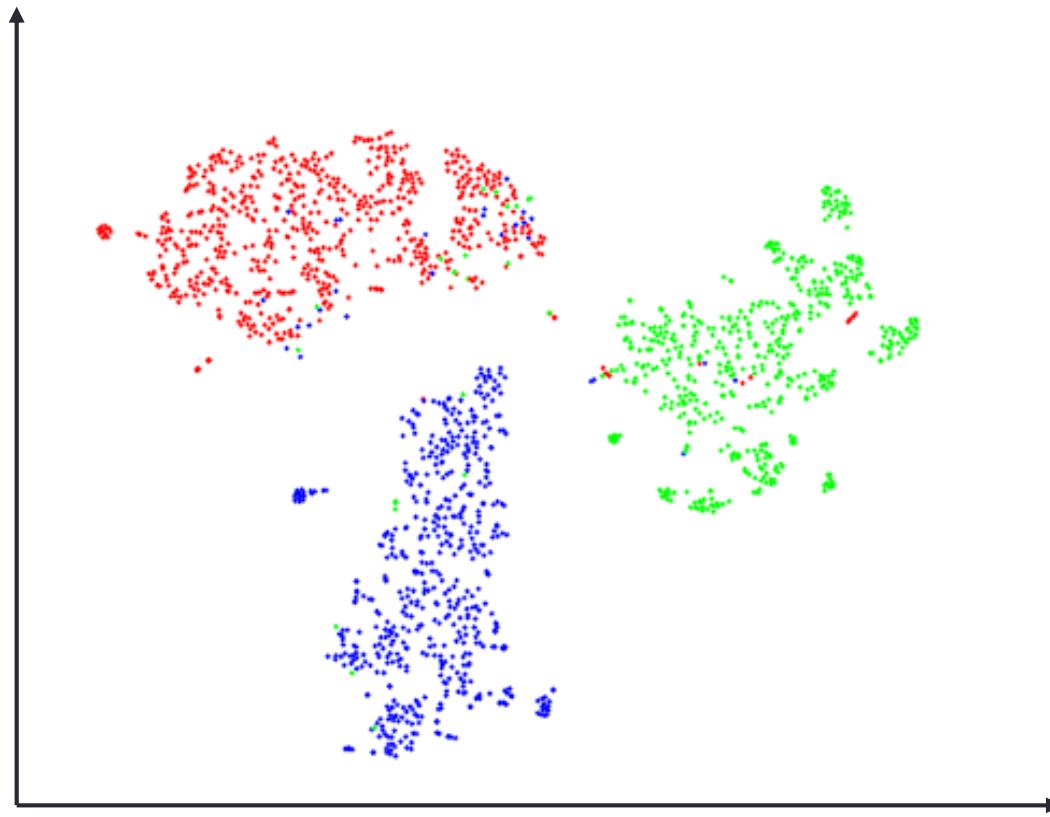
# Network Embedding



$G = (V)$   
Vector Space

- Easy to parallel
- Can apply classical ML methods

# The ultimate goal



## Network Inference

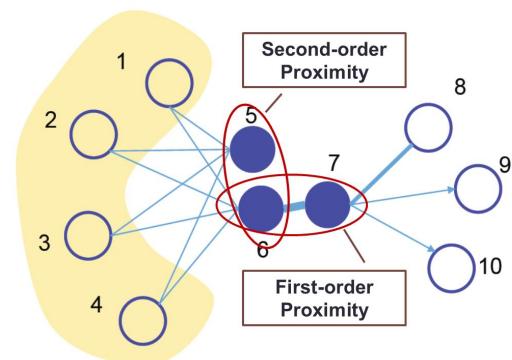
- Node importance
- Community detection
- Network distance
- Link prediction
- Node classification
- Network evolution
- ...

*in Vector Space*

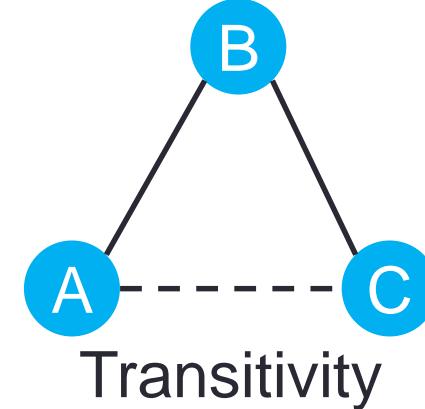
# The goal of network embedding

**Goal** Support network inference in vector space

Reflect network structure



Maintain network properties



Transform network nodes into vectors that are fit for off-the-shelf machine learning models

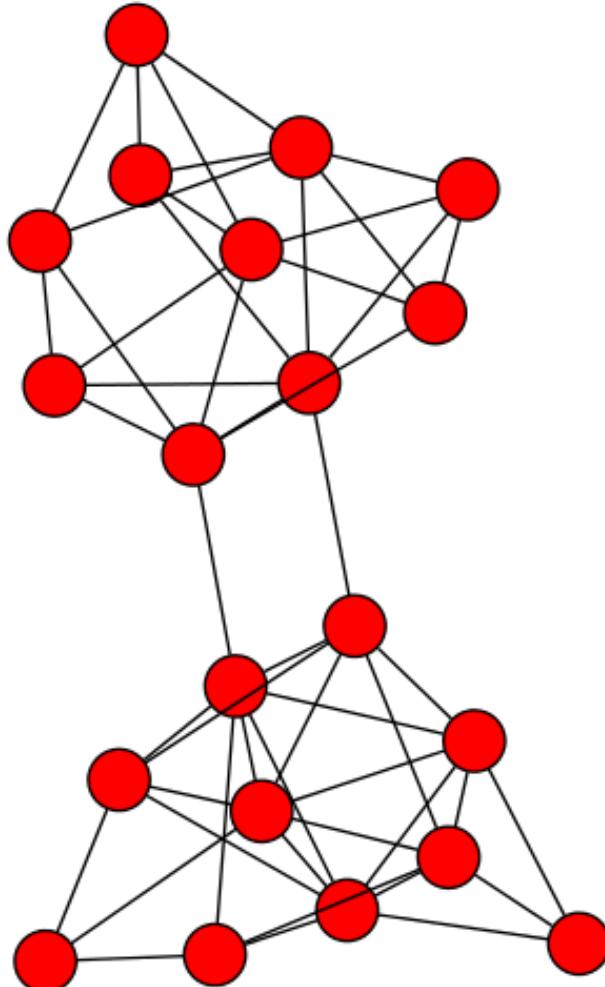
# Outline

- **Structure-preserved network embedding**
- **Property-preserved network embedding**
- **Dynamic network embedding**

# Outline

- **Structure-preserved network embedding**
- **Property-preserved network embedding**
- **Dynamic network embedding**

# Network Structures



Nodes & Links



Pair-wise Proximity



Community Structures



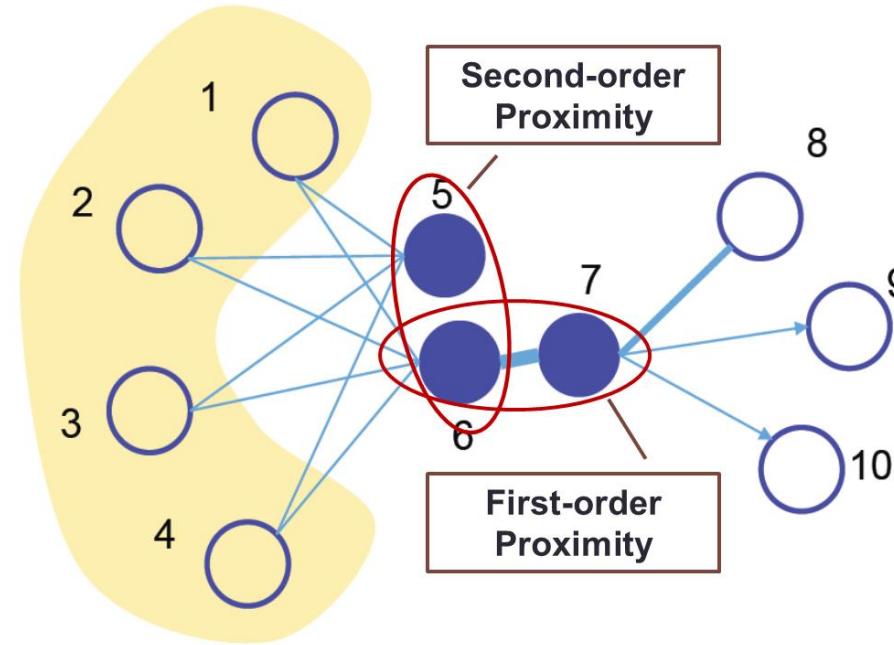
Hyper Edges



Global Structure

# High-Order Proximity

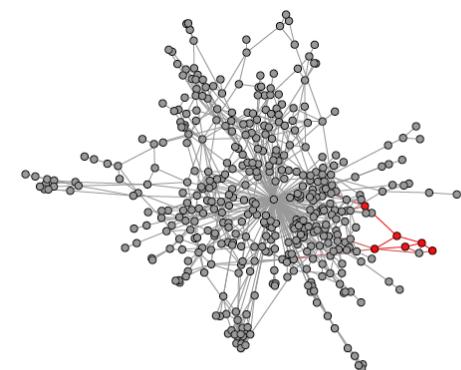
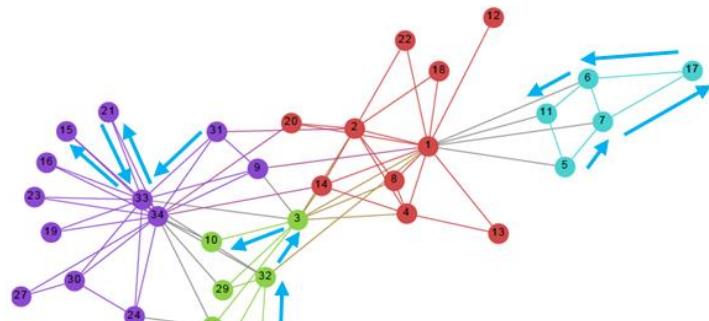
- Capturing the underlying structure of networks



- Advantages:
  - Solve the sparsity problem of network connections
  - Measure indirect relationship between nodes

# DeepWalk

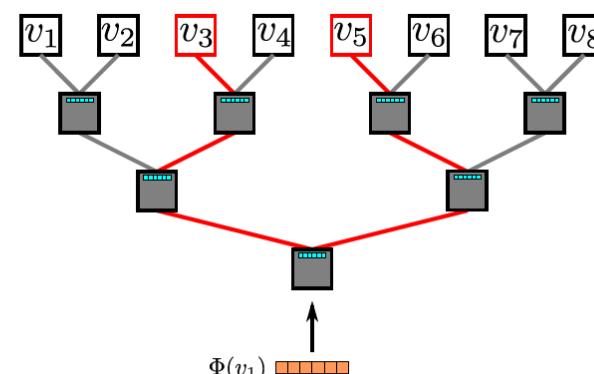
- Exploit truncated random walks to define neighborhoods of a node.



(a) Random walk generation.

$$\mathcal{W}_{v_4} = \begin{bmatrix} 3 \\ 1 \\ 5 \\ 1 \\ \vdots \end{bmatrix}_{v_j} \rightarrow \Phi^d_j$$

(b) Representation mapping.

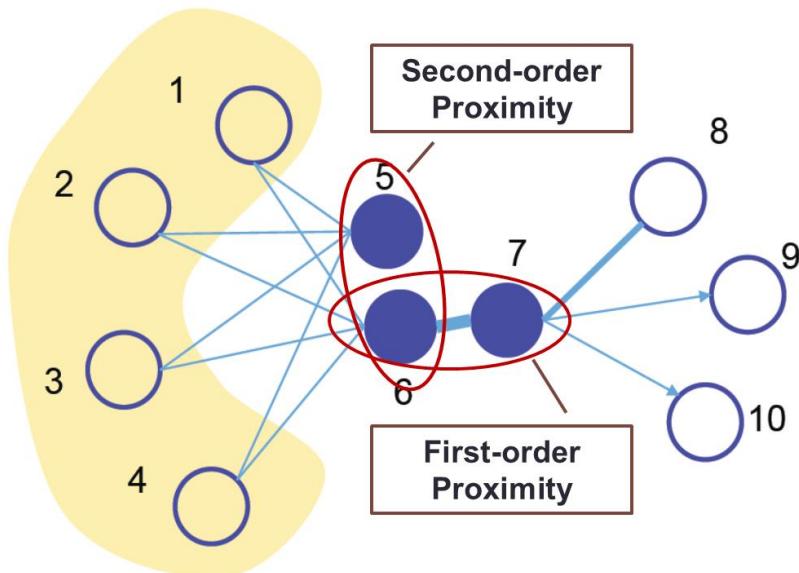


(c) Hierarchical Softmax.

## Random Walks on Graph

- $V_{26} - V_{25} - V_{32} - V_3 - V_{10} \dots$
- $V_5 - V_7 - V_{17} - V_6 - V_{11} \dots$
- $V_{31} - V_{33} - V_{21} - V_{33} - V_{15}$

# LINE



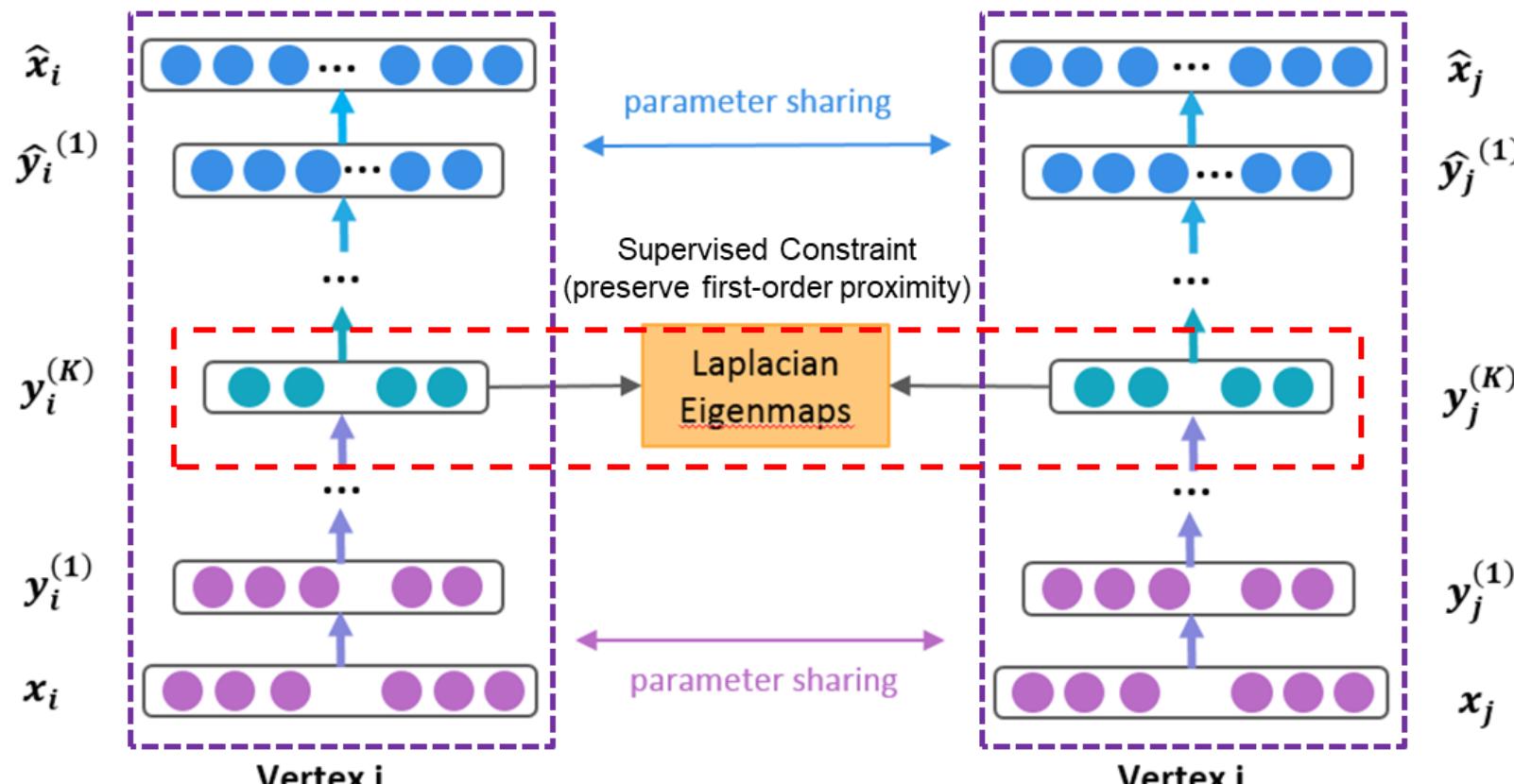
LINE with First-order Proximity:  
local pairwise

$$O_1 = - \sum_{(i,j) \in E} w_{ij} \log p_1(v_i, v_j)$$

LINE with Second-order Proximity:  
neighborhood structures

$$O_2 = \sum_{i \in V} \lambda_i d(\hat{p}_2(\cdot | v_i), p_2(\cdot | v_i))$$

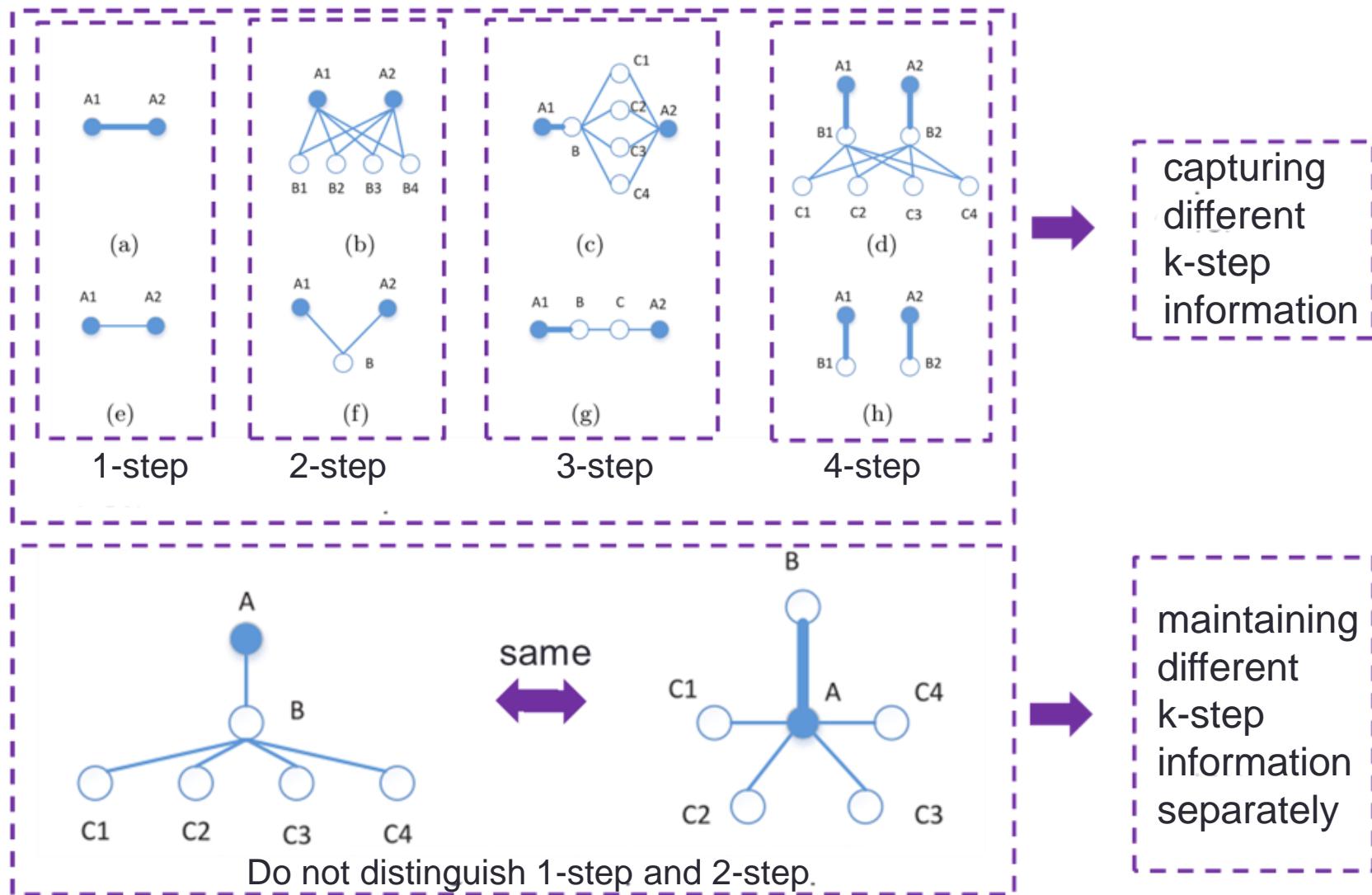
# SDNE – Structural Deep Network Embedding



Unsupervised Autoencoder  
(preserve second-order proximity)

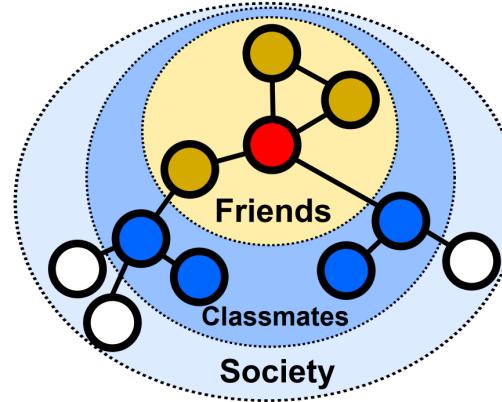
Unsupervised Autoencoder  
(preserve second-order proximity)

# GraRep



# What is the *right* order?

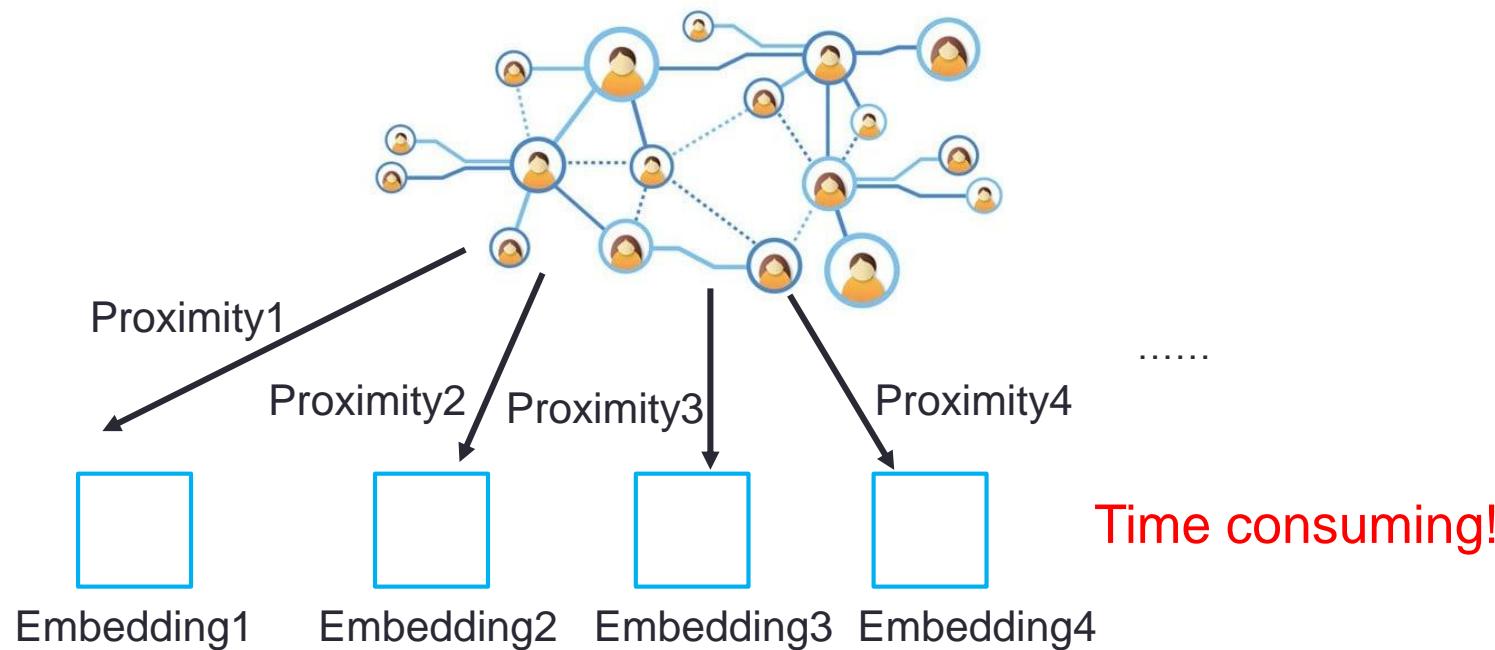
- Different networks/tasks require different high-order proximities
  - E.g., multi-scale classification (Bryan Perozzi, et al, 2017)



- E.g., networks with different scales and sparsity
- Proximities of different orders can also be arbitrarily weighted
  - E.g., equal weights, exponentially decayed weights (Katz)

# What is the *right* order?

- Existing methods can only preserve one fixed high-order proximity
- Different high-order proximities are calculated separately



→ How to preserve arbitrary-order proximity while guaranteeing accuracy and efficiency?

# Problem Formulation

- High-order proximity: a polynomial function of the adjacency matrix

$$S = f(A) = w_1 A^1 + w_2 A^2 + \cdots + w_q A^q$$

- $q$ : order;  $w_1 \dots w_q$ : weights, assuming to be non-negative
- $A$ : could be replaced by other variations (such as the Laplacian matrix)
- Objective function: matrix factorization

$$\min_{U^*, V^*} \|S - U^* {V^*}^T\|_F^2$$

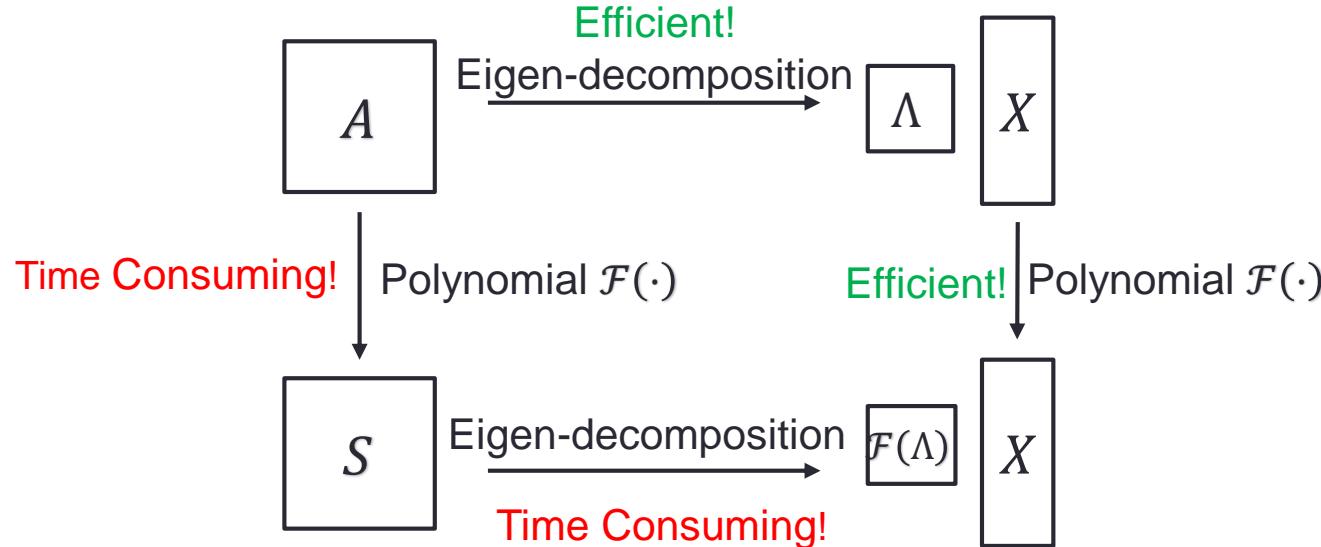
- $U^*, V^* \in \mathbb{R}^{N \times d}$ : left/right embedding vectors
- $d$ : dimensionality of the space
- Optimal solution: Singular Value Decomposition (SVD)
- $[U, \Sigma, V]$ : top-d SVD results

$$U^* = U\sqrt{\Sigma}, \quad V^* = V\sqrt{\Sigma}$$

# Eigen-decomposition Reweighting

## □ Eigen-decomposition reweighting

**THEOREM 4.2 (EIGEN-DECOMPOSITION REWEIGHTING).** *If  $[\lambda, \mathbf{x}]$  is an eigen-pair of  $\mathbf{A}$ , then  $[\mathcal{F}(\lambda), \mathbf{x}]$  is an eigen-pair of  $\mathbf{S} = \mathcal{F}(\mathbf{A})$ .*

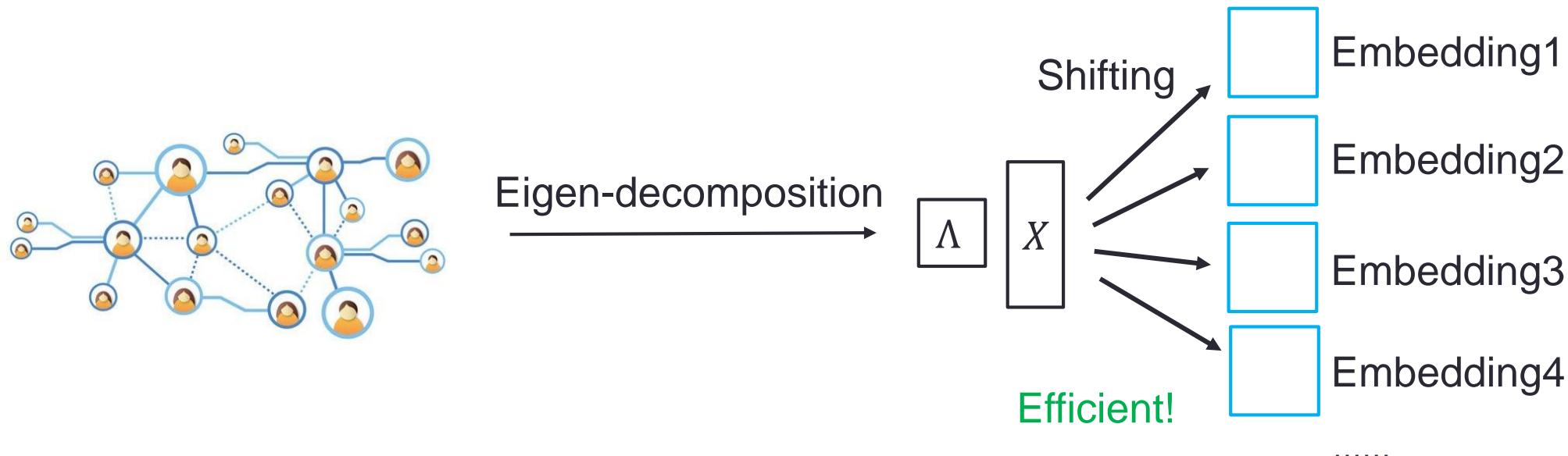


## □ Insights: high-order proximity is simply re-weighting dimensions!

$$U^* = U\sqrt{\Sigma}, V^* = V\sqrt{\Sigma}$$

# Preserving Arbitrary-Order Proximity

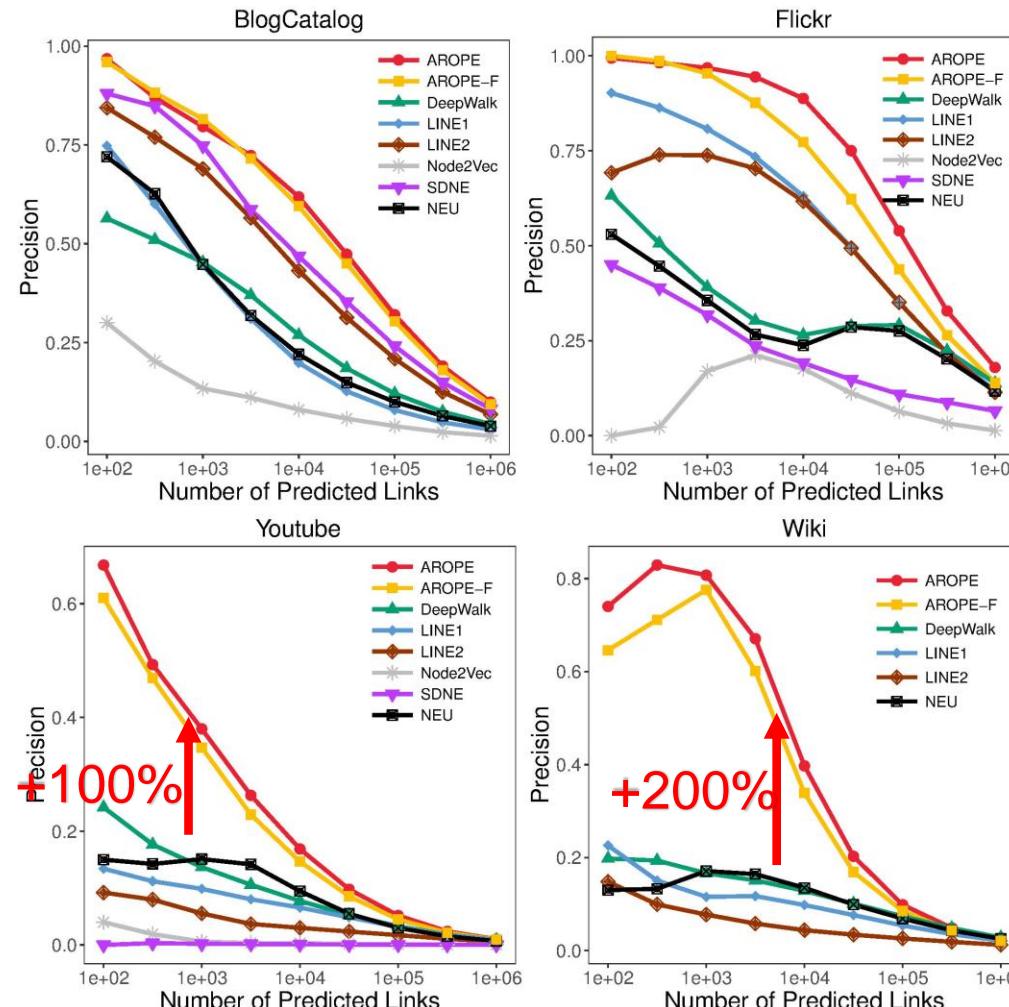
- Shifting across different orders/weights:



- Preserving arbitrary-order proximity
- Low marginal cost
- Accurate and efficient

# Experimental Results

## Link Prediction



# Billion-Scale Networks

- Existing network embeddings (e.g., AROPE) can handle million-scale networks
- But real graphs can have billions of nodes and edges
  - Social Networks
    - WeChat: 1 billion monthly active users (March, 2018)
    - Facebook: 2 billion active users (2017)
  - E-commerce Networks
    - Amazon: 353 million products, 310 million users, 5 billion orders (2017)
  - Citation Networks
    - 130 million authors, 233 million publications, 754 million citations (Aminer, 2018)



How to scale embedding methods to billion-scale networks?

# Random Projection for Matrix Factorization

- Objective function: matrix factorization of preserving high-order proximity

$$\min_{U,V} \|S - UV^T\|_F^2$$

$$S = f(A) = \alpha_1 A^1 + \alpha_2 A^2 + \cdots + \alpha_q A^q$$

- Essentially a dimensionality reduction problem
- Random projection: optimization-free for dimensionality reduction

- Basic idea: randomly project data into a low-dimensional subspace

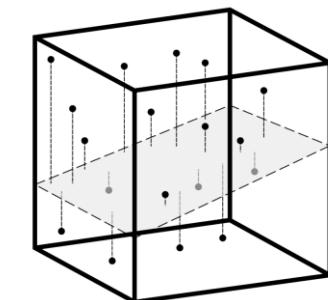
- Extremely efficient and friendly to distributed computing

- Denote  $R \in \mathbb{R}^{N \times d}$  as a Gaussian random matrix

$$R_{ij} \sim \mathcal{N}\left(0, \frac{1}{d}\right)$$

- Surprisingly simple result:

$$U = SR$$



# Theoretical Guarantee

- Slight modification: assuming positive semi-definite and using 2-norm

$$\min_U \|SS^T - UU^T\|_2$$

$$S = f(A) = \alpha_1 A^1 + \alpha_2 A^2 + \cdots + \alpha_q A^q$$

- Theoretical guarantee

**Theorem 1.** *For any similarity matrix  $\mathbf{S}$ , denote its rank as  $r_{\mathbf{S}}$ . Then, for any  $\epsilon \in (0, \frac{1}{2})$ , the following equation holds:*

$$P \left[ \|\mathbf{S} \cdot \mathbf{S}^T - \mathbf{U} \cdot \mathbf{U}^T\|_2 > \epsilon \|\mathbf{S}^T \cdot \mathbf{S}\|_2 \right] \leq 2r_{\mathbf{S}} e^{-\frac{(\epsilon^2 - \epsilon^3)d}{4}},$$

where  $\mathbf{U} = \mathbf{S} \cdot \mathbf{R}$  and  $\mathbf{R}$  is a Gaussian random matrix.

- Basically, random projection can effectively minimize the objective function
- However, calculating  $S$  is still very **time consuming**

# RandNE: Iterative Projection

- Iterative projection: can be calculated iteratively

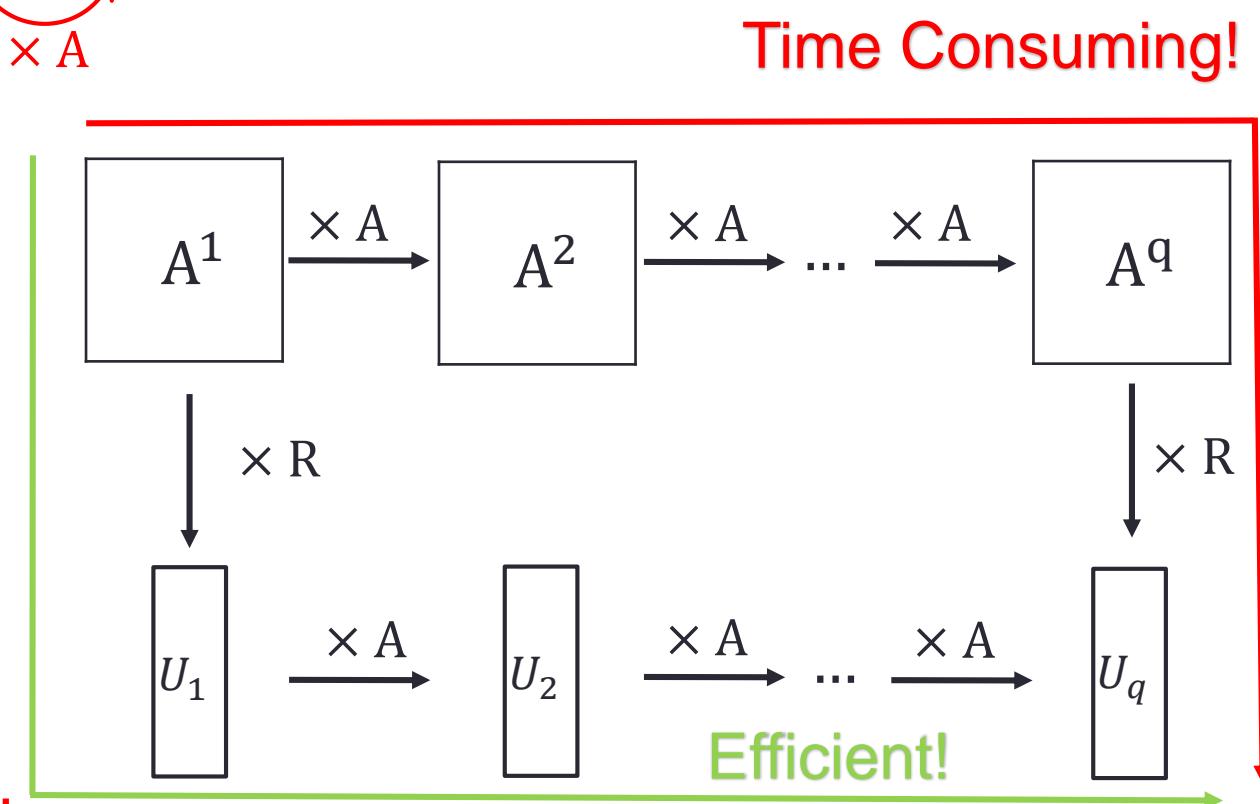
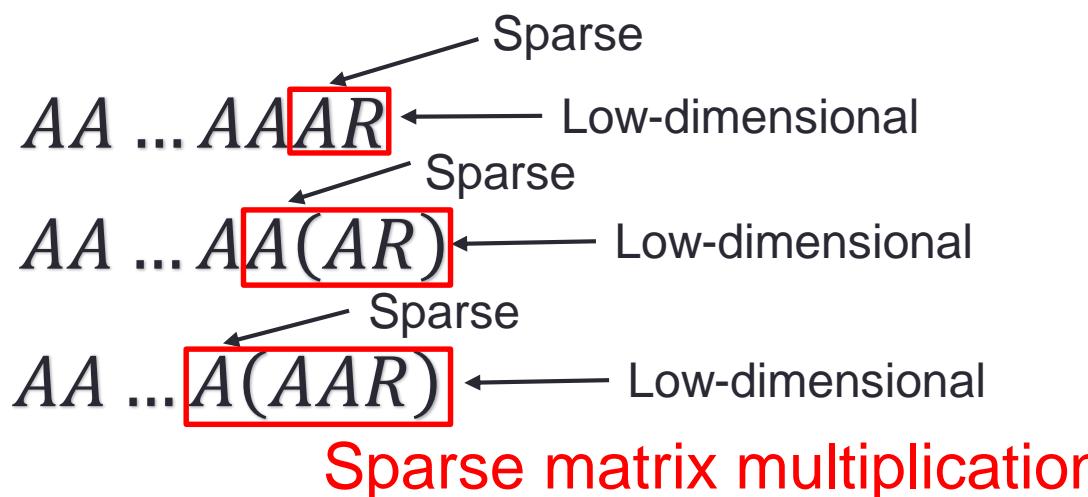
$$U = SR = (\alpha_1 A^1 + \alpha_2 A^2 + \dots + \alpha_q A^q)R$$

$$= \alpha_1 \boxed{A^1 R} + \alpha_2 \boxed{A^2 R} + \dots + \alpha_q \boxed{A^q R}$$

$\times A$        $\times A$        $\times A$

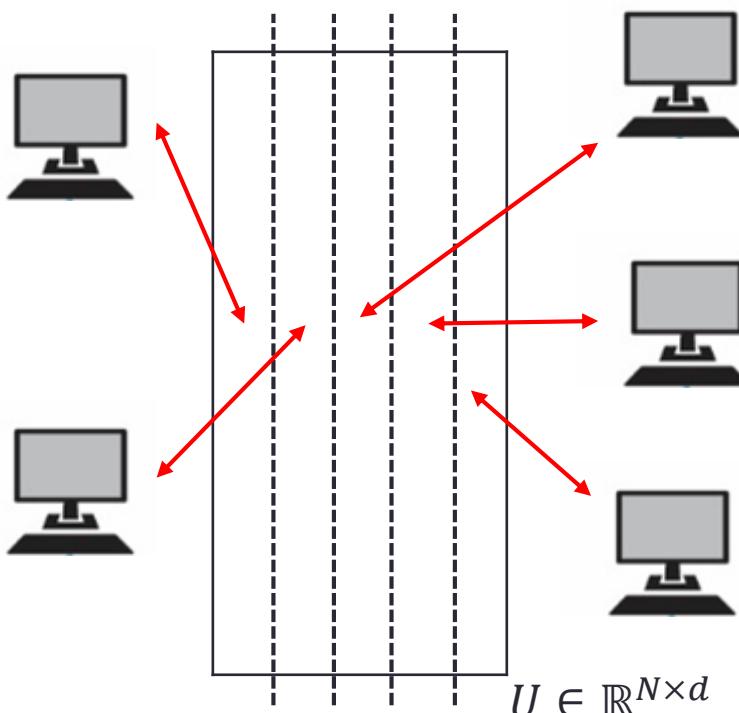
- Why efficient?

- $A: N \times N$  sparse adjacency matrix
- $R: N \times d$  low-dimensional matrix
- Associative law of matrix multiplication



# Distributed Calculation

- Iterative random projection only involves matrix multiplication  $U_i = AU_{i-1}$
- Each dimension can be calculated separately
  - Property of sparse matrix multiplication
- No communication is needed during calculation!




---

**Algorithm 2** Distributed Calculation of RandNE
 

---

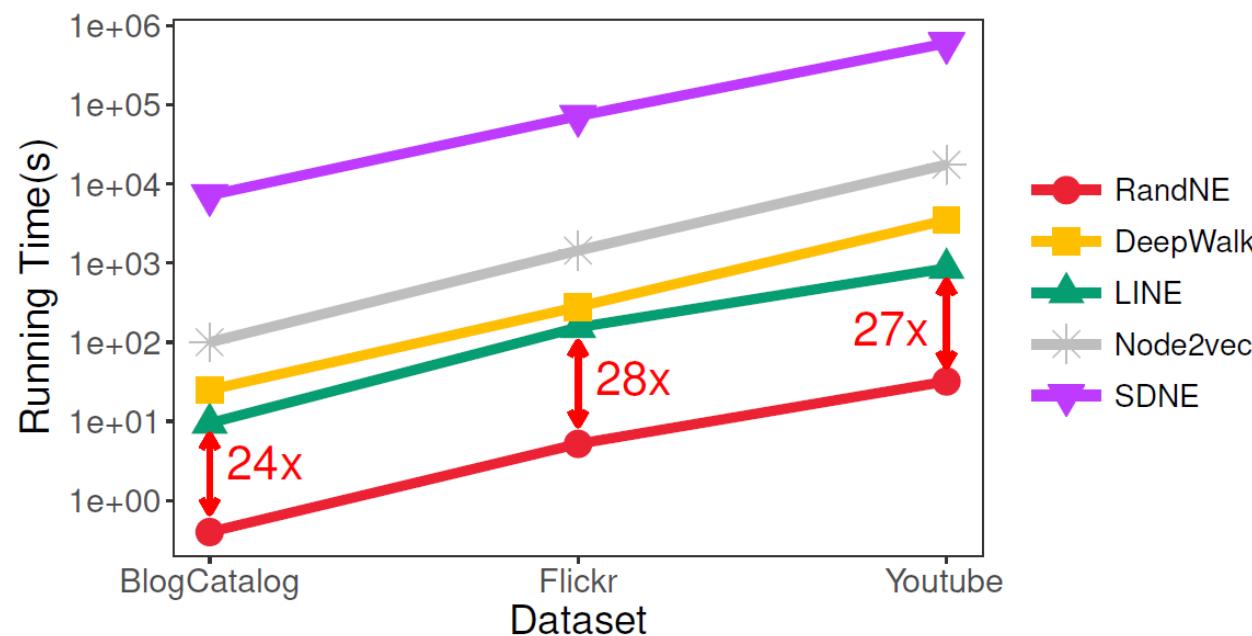
**Require:** Adjacency matrix  $\mathbf{A}$ , Initial Projection  $\mathbf{U}_0$ , Parameters of RandNE,  $K$  Distributed Servers

**Ensure:** Embedding Results  $\mathbf{U}$

- 1: Broadcast  $\mathbf{A}$ ,  $\mathbf{U}_0$  and parameters into  $K$  servers
  - 2: Set  $i = 1$
  - 3: **repeat**
  - 4:   **if** There is an idle server  $k$  **then**
  - 5:     Calculate  $\mathbf{U}(i, :)$  in server  $k$
  - 6:      $i = i + 1$
  - 7:     Gather  $\mathbf{U}(i, :)$  from server  $k$  after calculation
  - 8:   **end if**
  - 9: **until**  $i > d$
  - 10: Return  $\mathbf{U}$
-

# Experimental Results

## □ Running time



At least **dozens of times** faster

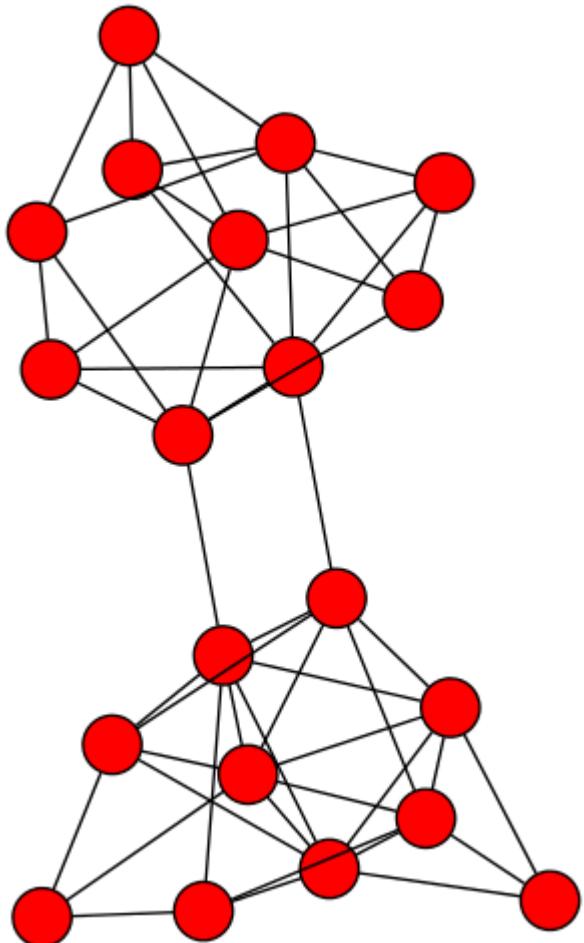
## □ Link Prediction

AUC SCORES OF LINK PREDICTION.

Dataset	BlogCatalog	Flickr	Youtube
RandNE	<b>0.944</b>	<b>0.940</b>	0.887
DeepWalk	0.760	0.938	0.909
LINE <sub>1st</sub>	0.667	0.909	0.847
LINE <sub>2nd</sub>	0.762	0.932	<b>0.959</b>
Node2vec	0.650	0.865	0.778
SDNE	0.940	0.926	-

Superior or comparable performance

# Section Summary



**Nodes & Links**



**Pair-wise Proximity**



**Community Structures**



**Hyper Edges**



**Global Structure**

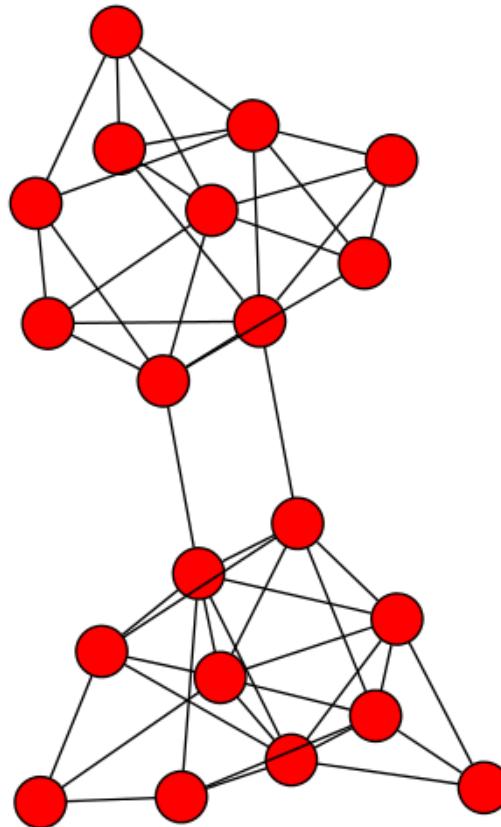
**Network  
Characteristics**

**Application  
Characteristics**

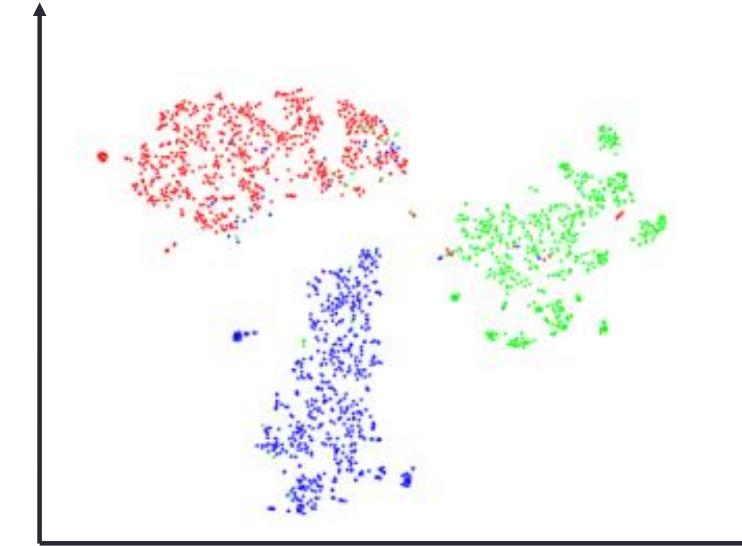
# Outline

- **Structure-preserved network embedding**
- **Property-preserved network embedding**
- **Dynamic network embedding**

# Why preserve network properties?



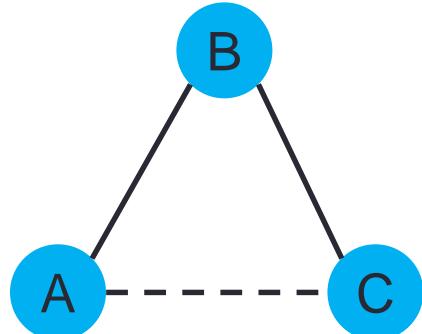
Heterogeneity



# Transitivity

## The Transitivity Phenomenon

**Network**



**Embedding Space**

**Triangle Inequality:**  $D(A, B) + D(B, C) > D(A, C)$

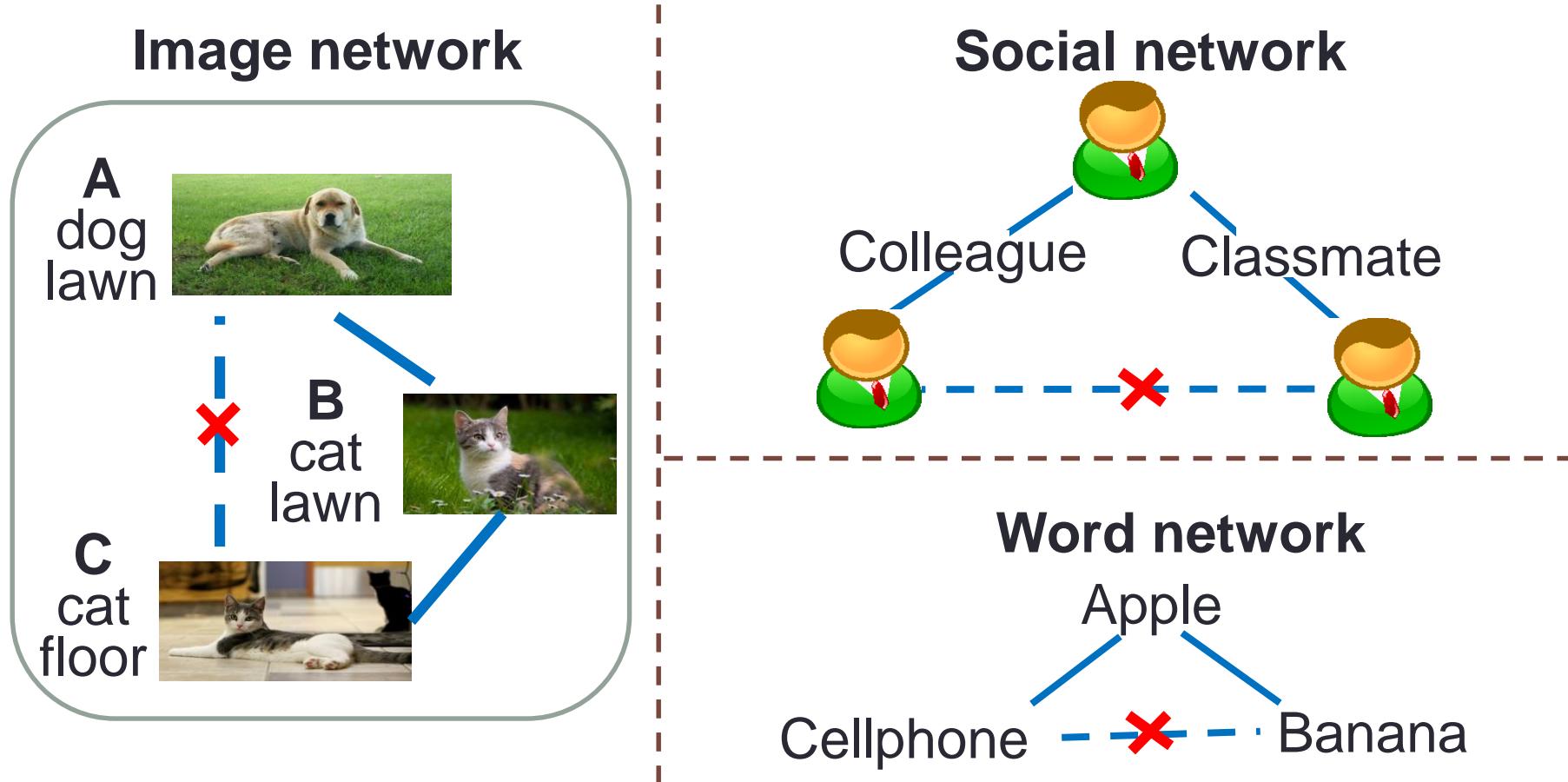


$A$  close to  $B$ ,  $B$  close to  $C \rightarrow A$  relatively close to  $C$

However, real network data is complex...

# Non-transitivity

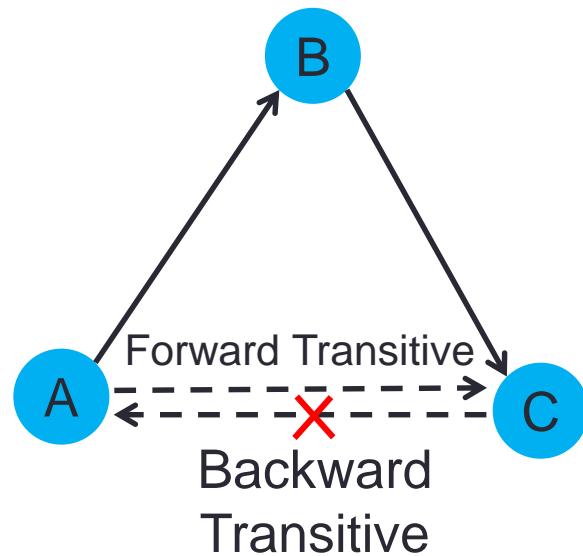
The Co-existence of *Transitivity* and *Non-transitivity*



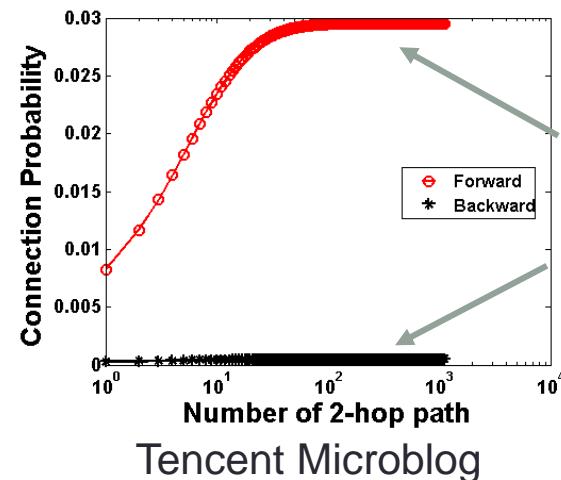
How to incorporate non-transitivity in the embedding space?

# Asymmetric Transitivity

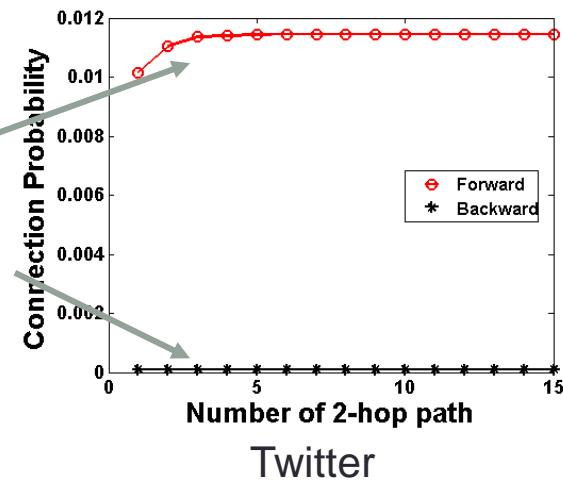
Directed Network



$A \rightarrow B, B \rightarrow C \Rightarrow A \rightarrow C$ , but not  $C \rightarrow A$



Forward  
Backward



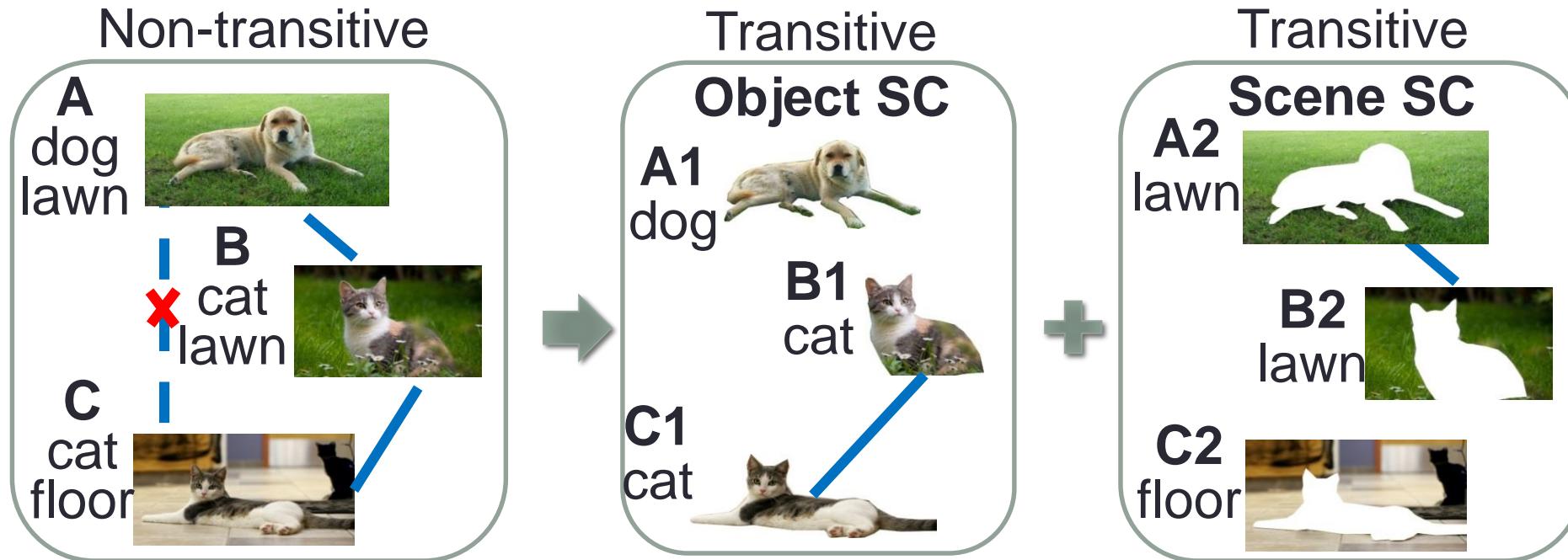
Twitter

Distance metric in the embedding space is symmetric.  
How to incorporate *Asymmetric Transitivity*?

# Non-transitivity

The source of non-transitivity:

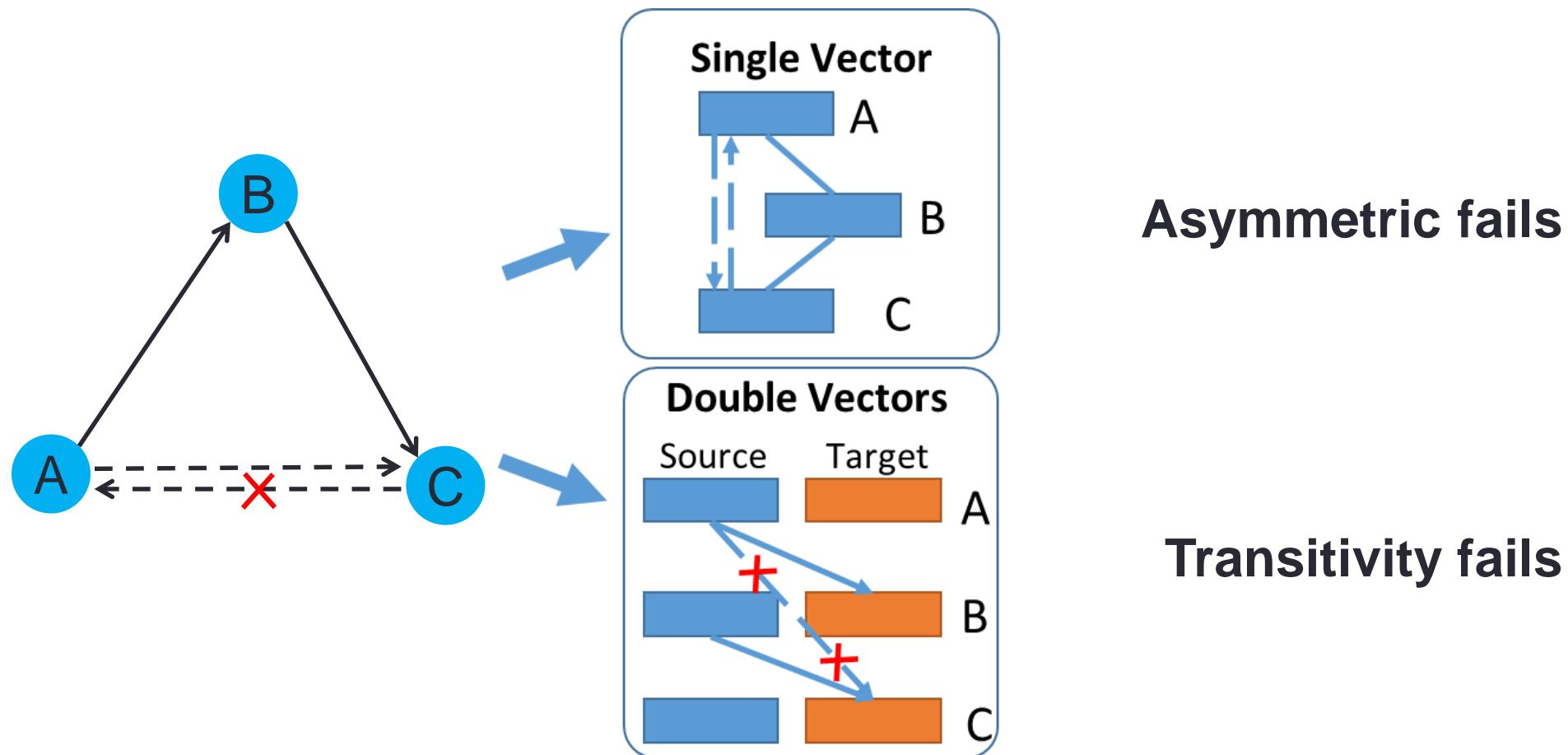
***Each node has multiple similarity components***



**Non-transitive Embedding:** represent non-transitive data with multiple latent similarity components

# Asymmetric Transitivity

All existing methods fail..



## Section Summary

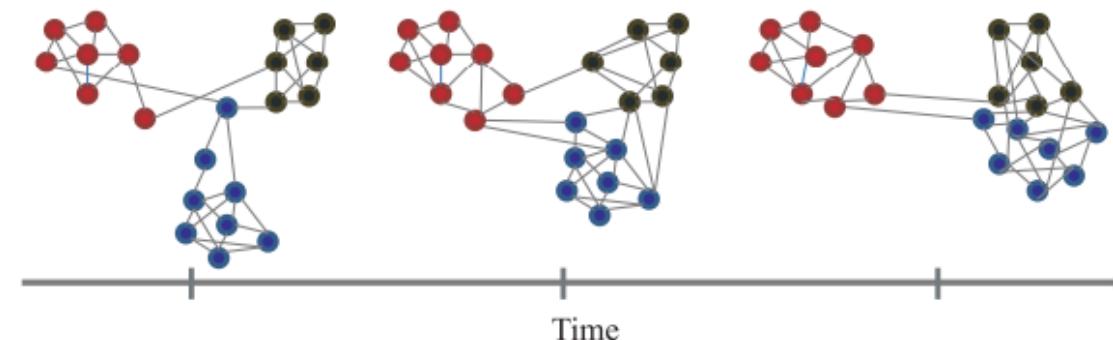
- Compared with network structures, **network properties** have a large space to explore in network embedding
- Transitivity is important for network inference.
- Uncertainty provides evidence in making network inference.
- Many other property issues:
  - The right embedding space: Euclidean space?
  - Power-law distribution
  - ...

# Outline

- **Structure-preserved network embedding**
- **Property-preserved network embedding**
- **Dynamic network embedding**

# Dynamic Networks

- Networks are dynamic in nature
  - New (old) nodes are added (deleted)
    - New users, products, etc.
  - The edges between nodes evolve over time
    - Users add or delete friends in social networks, or neurons establish new connections in brain networks.
- How to efficiently incorporate the dynamic changes when networks evolve?

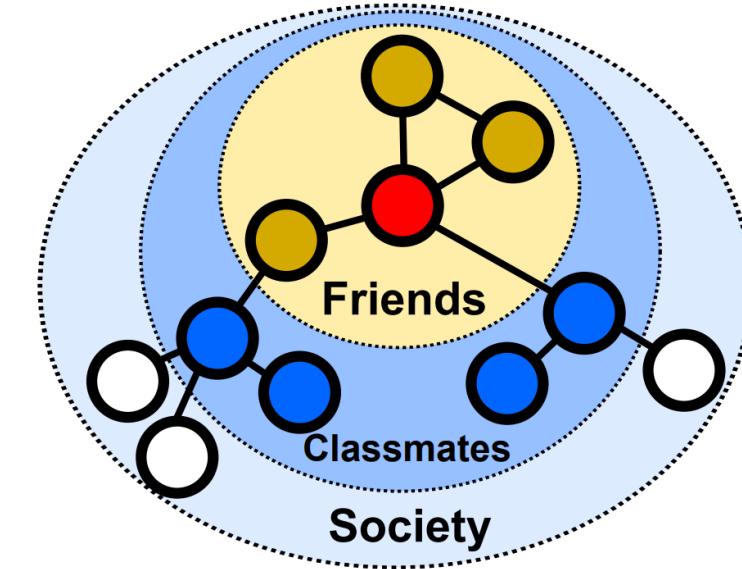


# Key problems in dynamic network embedding

- I : Out-of-sample nodes
- II : Incremental edges
- III: Aggregated error
- IV: Scalable optimization

# Challenge: High-order Proximity

- **High-order proximity**
  - Critical structural property of networks
  - Measure indirect relationship between nodes
  - Capture the structure of networks with different scales and sparsity



*Network Embedding vs. Traditional Graph Embedding*

# Challenge: High-order Proximity

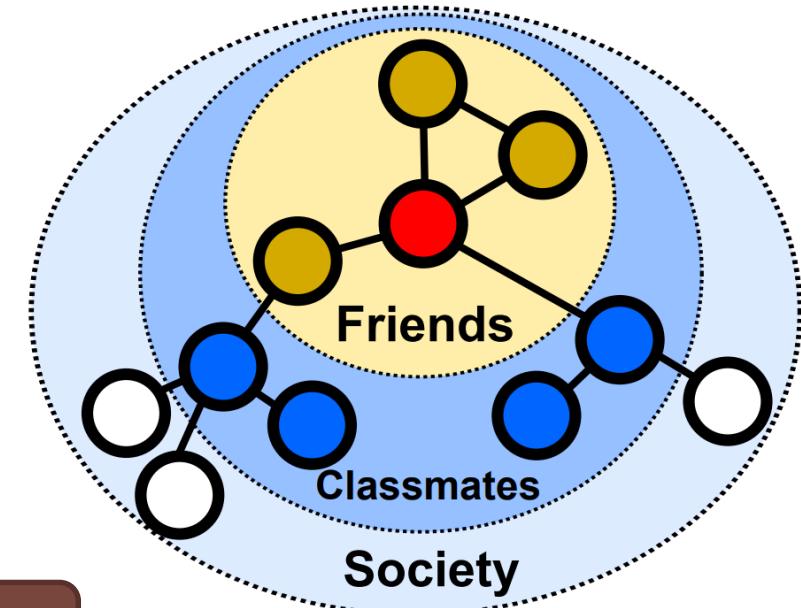
- I : Out-of-sample nodes
- II : Incremental edges
- III: Aggregated error
- IV: Scalable optimization



*Preserve High-order Proximities*



***Local Change leads to Global Updating***

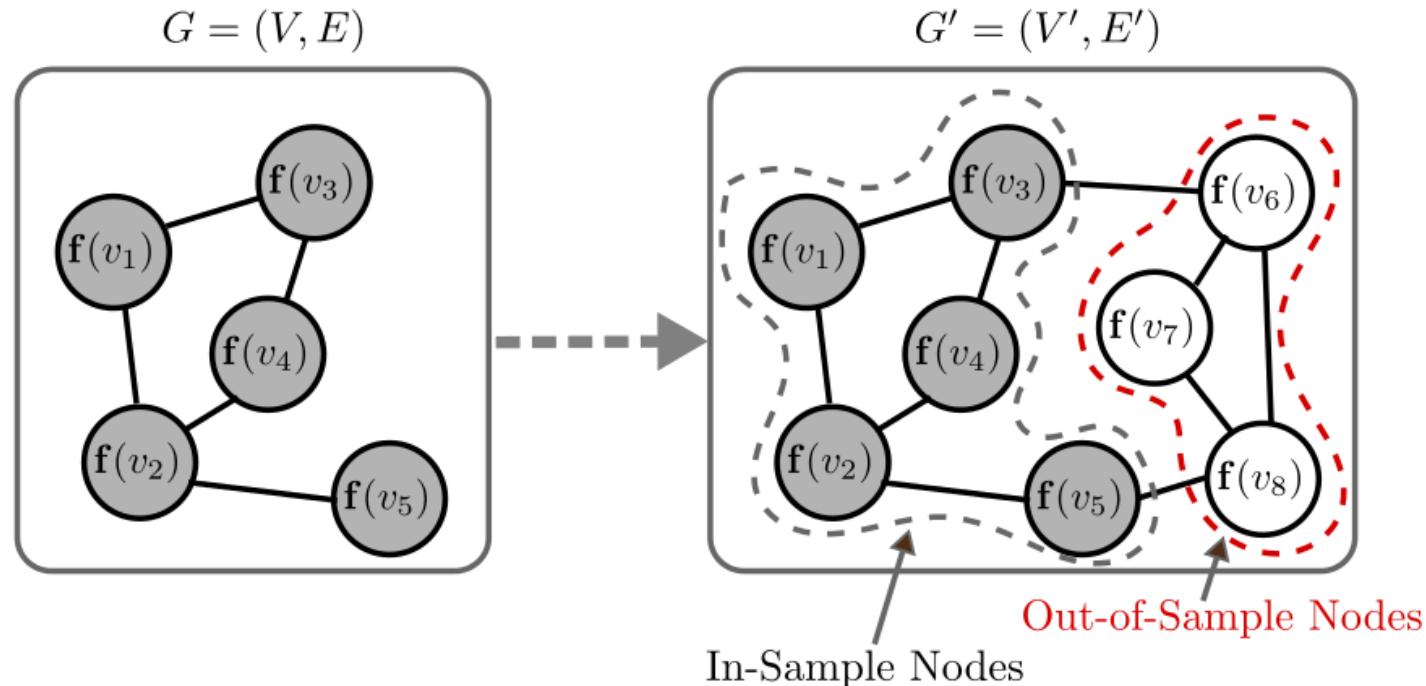


# Key problems in dynamic network embedding

- I : Out-of-sample nodes
- II : Incremental edges
- III: Aggregated error
- IV: Scalable optimization

# Problem

- To infer embeddings for out-of-sample nodes



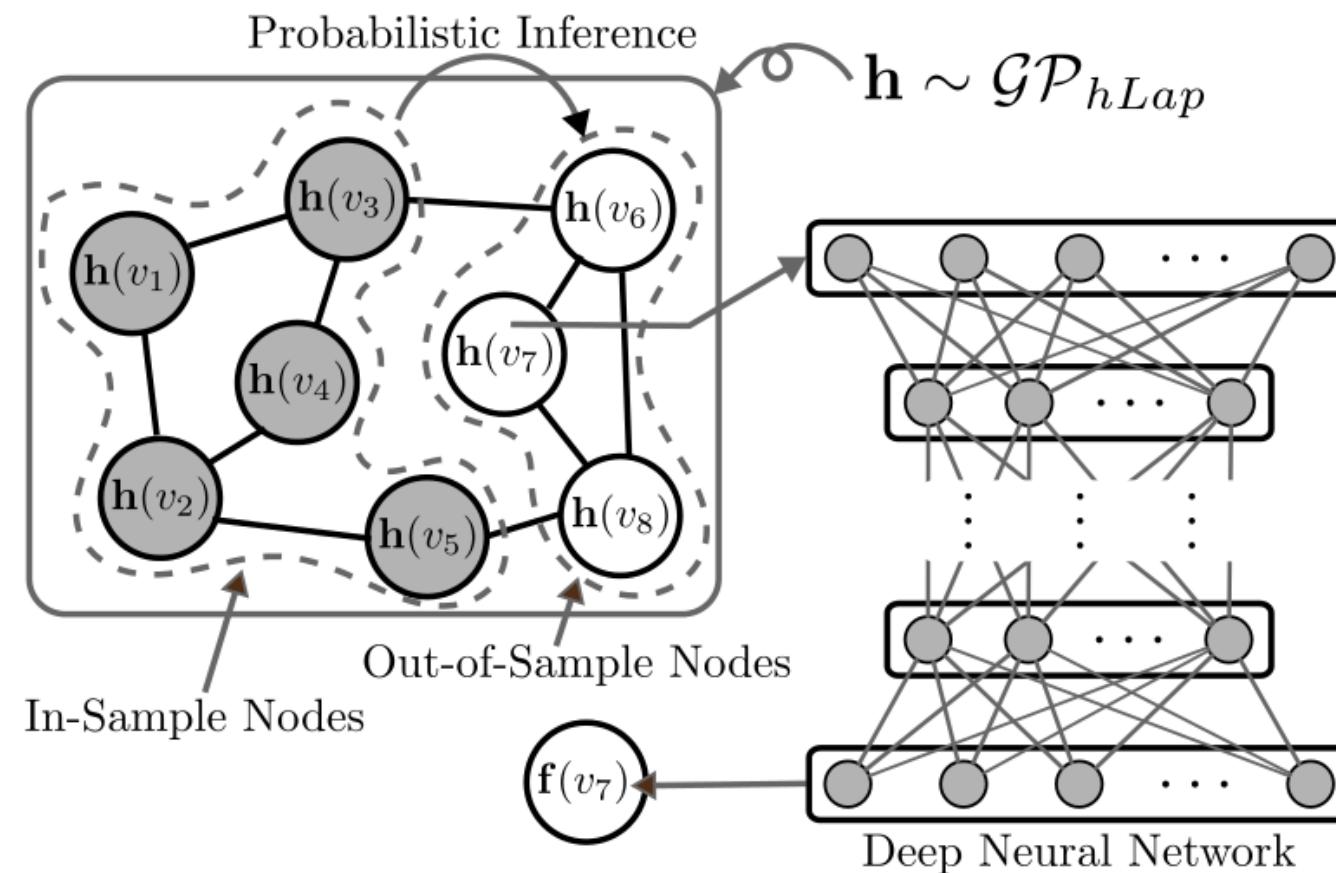
- $G=(V, E)$  evolves into  $G'=(V', E')$ , where  $V' = V \cup V^*$ .
- $n$  old nodes:  $V = \{v_1, \dots, v_n\}$ ,  $m$  new nodes:  $V^* = \{v_{n+1}, \dots, v_{n+m}\}$
- Network embedding  $f: V \rightarrow \mathbb{R}^d$
- We know  $f(V)$  for old nodes, want to infer  $f(V')$  for new nodes.

# Challenges

- Preserve network structures
  - E.g., high-order proximity
  - Need to incorporate prior knowledge on networks
- Share similar characteristics with in-sample embeddings
  - E.g. magnitude, mean, variance
  - Requires a model with great expressive power to fit the data well
- Low computational cost

# DepthLGP

- Nonparametric probabilistic modeling + Deep Learning



# DepthLGP

- Design a kernel for the  $k$ th ( $k=1, \dots, s$ ) dimension of  $h(\cdot)$

$$\mathbf{K}_k \triangleq \left[ \mathbf{I} + \boxed{\eta_k \mathbf{L}(\hat{\mathbf{A}}_k)} + \boxed{\zeta_k \mathbf{L}(\hat{\mathbf{A}}_k \hat{\mathbf{A}}_k)} \right]^{-1},$$

First-order Proximity      Second-order Proximity

$$\hat{\mathbf{A}}_k \triangleq \text{diag}(\boldsymbol{\alpha}_k) \mathbf{A}' \text{diag}(\boldsymbol{\alpha}_k),$$

$$\boldsymbol{\alpha}_k \triangleq \boxed{[a_{v_1}^{(k)}, a_{v_2}^{(k)}, \dots, a_{v_{n+m}}^{(k)}]}^\top,$$

Node Weights  
(to prune uninformative nodes)

<sup>1</sup>The matrix inversion can be bypassed without approximation.

<sup>2</sup> $a_v^{(k)}$  indicates how much attention we pay to a node. It is learned for an in-sample node, but fixed to one for an out-of-sample node, as we are always interested in out-of-sample nodes.

# Experimental Results: Classification

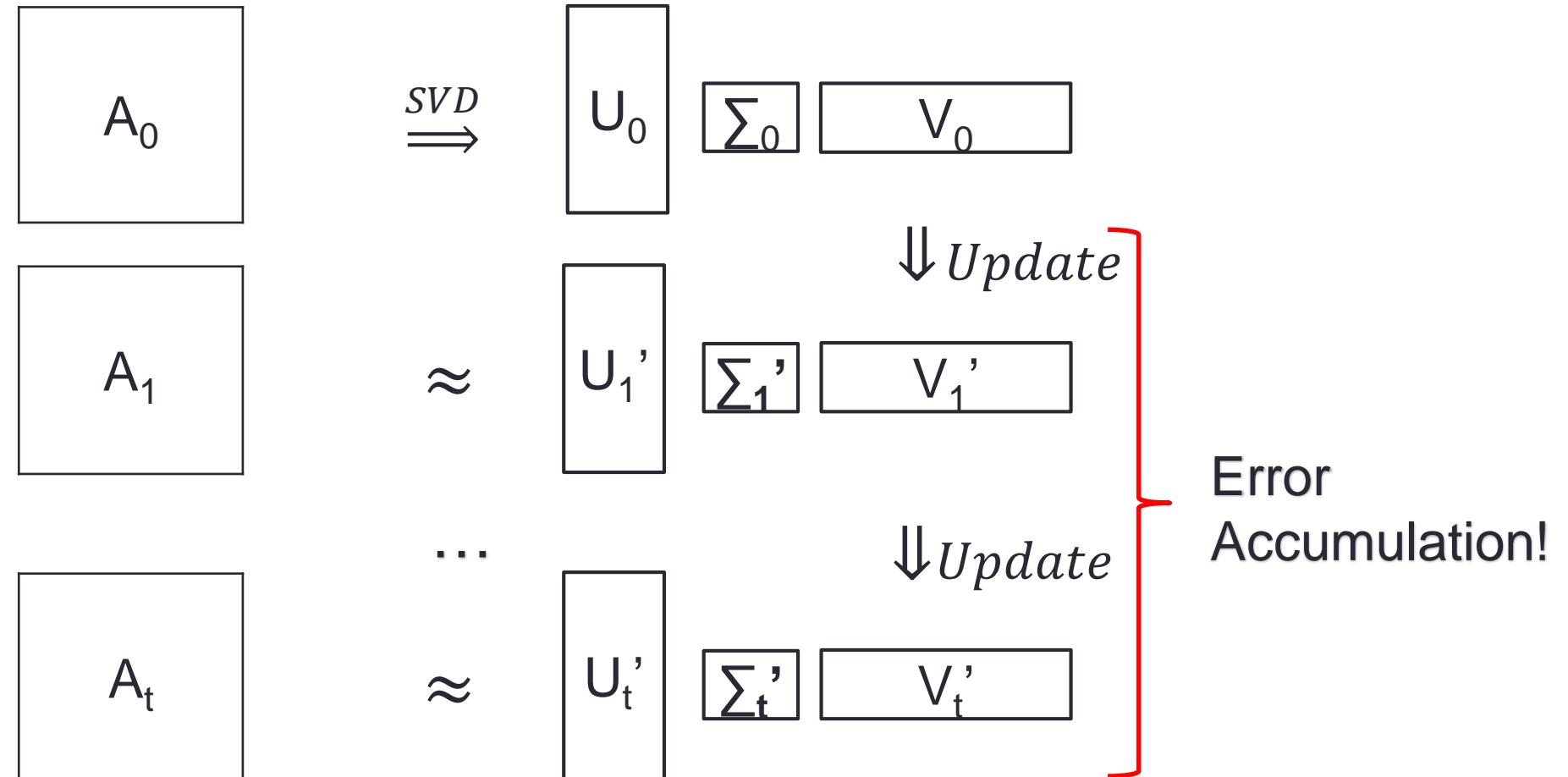
Metric	Embedding	Network	Baselines			This Work		Upper Bound (rerunning)
			LocalAvg	MRG	LabelProp	hLGP	DepthLGP	
Macro-F1(%)	LINE	DBLP	37.89	42.15	40.83	47.33	<b>48.25</b>	(49.07)
		PPI	10.52	10.02	12.42	13.42	<b>13.72</b>	(13.91)
		BlogCatalog	13.25	11.30	17.07	17.41	<b>18.03</b>	(18.90)
	GraRep	DBLP	50.61	55.79	55.02	57.43	<b>58.67</b>	(62.92)
		PPI	13.65	13.75	12.38	14.80	<b>14.84</b>	(15.33)
	node2vec	BlogCatalog	14.76	14.80	14.71	15.94	<b>18.45</b>	(20.15)
		DBLP	53.83	59.34	59.25	60.89	<b>62.63</b>	(64.87)
		PPI	15.05	13.43	13.78	15.85	<b>16.54</b>	(16.81)
		BlogCatalog	15.10	14.04	19.16	19.77	<b>20.32</b>	(20.82)
Micro-F1(%)	LINE	DBLP	49.58	50.49	50.88	54.01	<b>54.94</b>	(55.84)
		PPI	18.10	15.71	18.81	20.71	<b>21.42</b>	(21.43)
		BlogCatalog	27.40	23.21	30.79	31.36	<b>31.90</b>	(32.20)
	GraRep	DBLP	60.17	60.62	60.48	61.44	<b>62.29</b>	(65.44)
		PPI	20.23	20.35	20.23	20.79	<b>21.44</b>	(21.88)
	node2vec	BlogCatalog	36.44	30.79	33.90	37.57	<b>38.14</b>	(38.37)
		DBLP	60.54	62.29	62.52	62.83	<b>64.56</b>	(65.63)
		PPI	19.70	18.25	18.25	22.63	<b>23.11</b>	(23.41)
		BlogCatalog	34.83	25.82	36.94	37.96	<b>39.64</b>	(40.34)

# Key problems in dynamic network embedding

- I : Out-of-sample nodes
- II : Incremental edges
- III: Aggregated error
- IV: Scalable optimization

# Problem: Error Accumulation

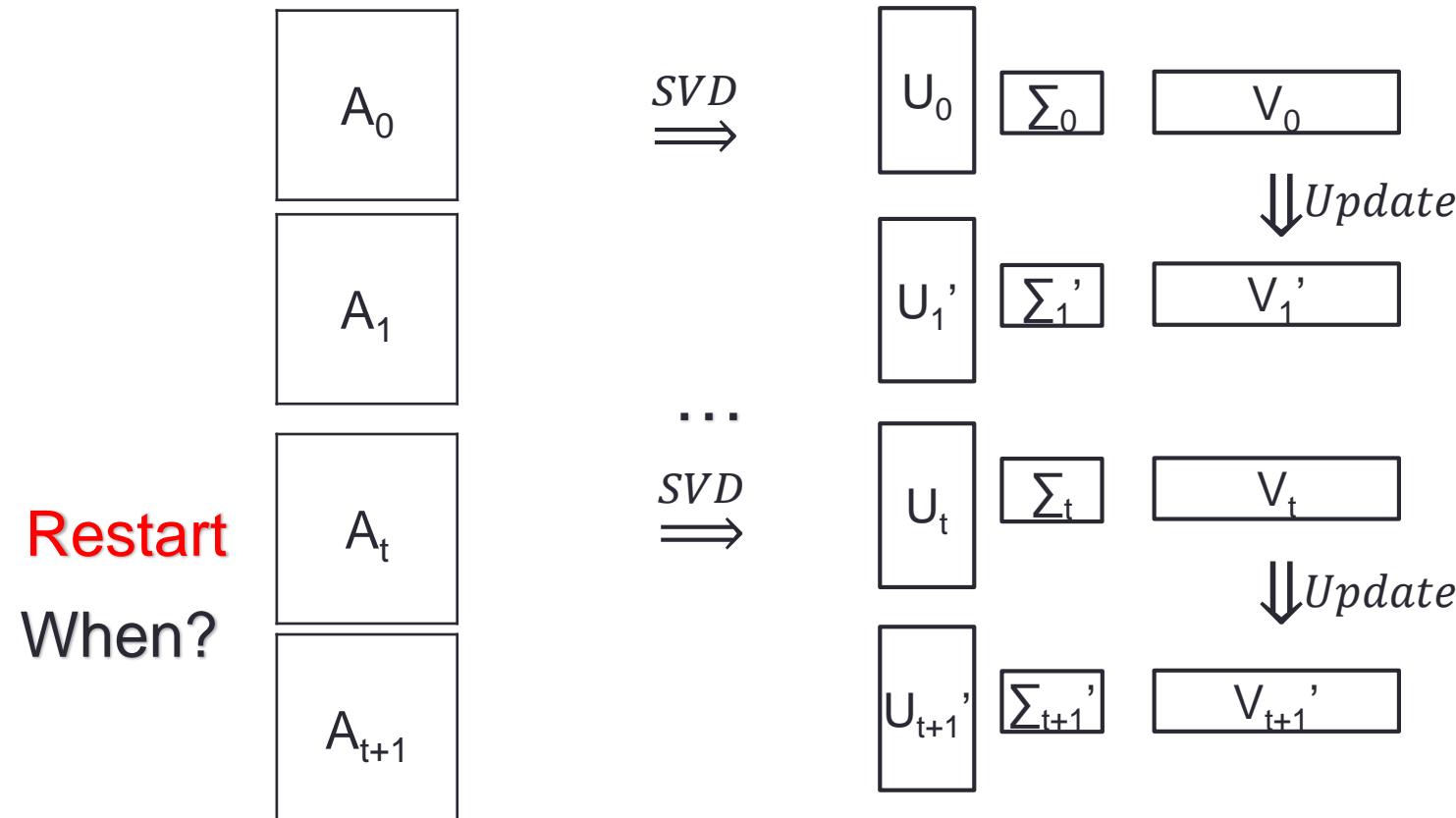
- Eigen perturbation is at the cost of inducing approximation



- Problem: error accumulation is inevitable

# Solution: SVD Restarts

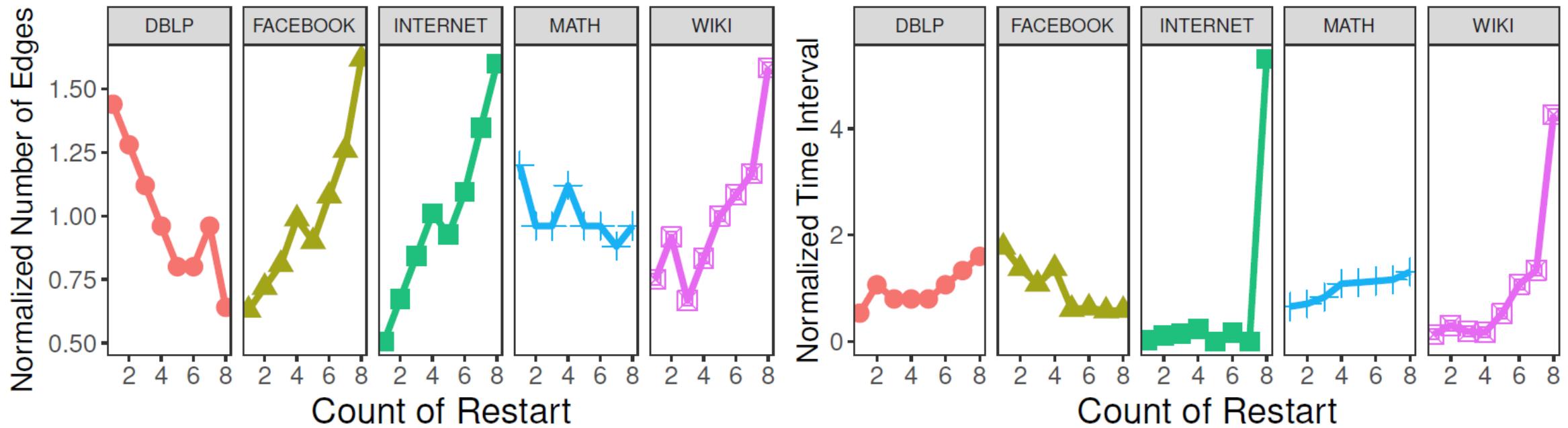
- Solution: restart SVD occasionally



- What are the appropriate time points?
  - Too early restarts: waste of computation resources
  - Too late restarts: serious error accumulation

# Naïve Solution

- Naïve solution: fixed time interval or fixed number of changes
- Difficulty: error accumulation is not uniform



# Existing Method

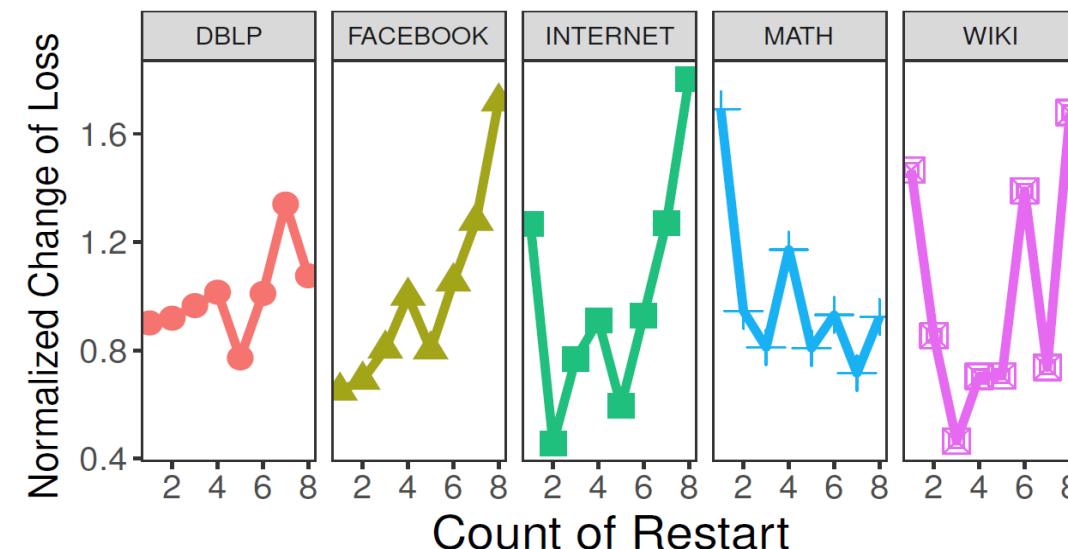
- Existing method: monitor loss (Chen and Candan, KDD 2014)

- Loss in SVD:

$$\mathcal{J} = \|S - U\Sigma V^T\|_F^2$$

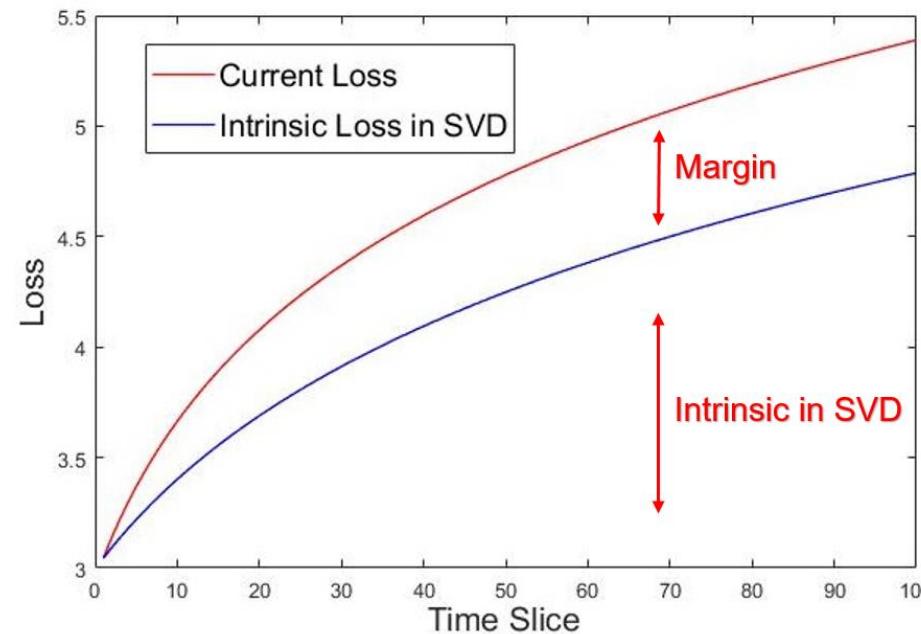
$S$ : target matrix,  $[U, \Sigma, V]$ : results of SVD

- Problem: loss includes approximation error and intrinsic loss in SVD



# Framework: Monitor Margin

- Observation: the margin between the current loss and the intrinsic loss in SVD is the actual accumulated error
  - Current loss:  $\mathcal{J} = \|S - U\Sigma V^T\|_F^2$
  - Intrinsic loss:  $\mathcal{L}(S, k) = \min_{U^*, \Sigma^*, V^*} \|S - U^* \Sigma^* V^{*T}\|_F^2, k: \text{dimensionality}$



# Solution: Lazy Restarts

- Lazy restarts: restart only when the margin exceeds the threshold
- Problem: intrinsic loss is hard to compute
  - Direct calculation has the same time complexity as SVD
- Relaxation: an upper bound on margin
  - A lower bound on intrinsic loss  $\mathcal{L}(S, k)$

$$\mathcal{L}(\mathbf{S}_t, k) \geq B(t) \Rightarrow \frac{\mathcal{J}(t) - \mathcal{L}(\mathbf{S}_t, k)}{\mathcal{L}(\mathbf{S}_t, k)} \leq \frac{\mathcal{J}(t) - B(t)}{B(t)}.$$

$\mathcal{J}(t)$ : current loss;  $\mathcal{L}(S_t, k)$ : intrinsic loss;  $B(t)$ : bound of intrinsic loss

# A Lower Bound of SVD Intrinsic Loss

- Idea: use matrix perturbation

**Theorem 1** (A Lower Bound of SVD Intrinsic Loss). *If  $\mathbf{S}$  and  $\Delta\mathbf{S}$  are symmetric matrices, then:*

$$\mathcal{L}(\mathbf{S} + \Delta\mathbf{S}, k) \geq \mathcal{L}(\mathbf{S}, k) + \Delta\text{tr}^2(\mathbf{S} + \Delta\mathbf{S}, \mathbf{S}) - \sum_{l=1}^k \lambda_l, \quad (9)$$

where  $\lambda_1 \geq \lambda_2 \dots \geq \lambda_k$  are the top- $k$  eigenvalues of  $\nabla_{\mathbf{S}^2} = \mathbf{S} \cdot \Delta\mathbf{S} + \Delta\mathbf{S} \cdot \mathbf{S} + \Delta\mathbf{S} \cdot \Delta\mathbf{S}$ , and

$$\Delta\text{tr}^2(\mathbf{S} + \Delta\mathbf{S}, \mathbf{S}) = \text{tr}((\mathbf{S} + \Delta\mathbf{S}) \cdot (\mathbf{S} + \Delta\mathbf{S})) - \text{tr}(\mathbf{S} \cdot \mathbf{S}).$$

- Intuition: treat changes as a perturbation to the original network

# Time Complexity Analysis

**Theorem 2.** *The time complexity of calculating  $B(t)$  in Eqn (13) is  $O(M_S + M_L k + N_L k^2)$ , where  $M_S$  is the number of the non-zero elements in  $\Delta\mathbf{S}$ , and  $N_L, M_L$  are the number of the non-zero rows and elements in  $\nabla_{S^2}$  respectively.*

- If every node has a equal probability of adding new edges, we have:  $M_L \approx 2d_{avg}M_S$ , where  $d_{avg}$  is the average degree of the network .
- For Barabasi Albert model (Barabási and Albert 1999), a typical example of preferential attachment networks, we have:  $M_L \approx \frac{12}{\pi^2} [\log(d_{max}) + \gamma] M_S$ , where  $d_{max}$  is the maximum degree of the network and  $\gamma \approx 0.58$  is a constant.

- Conclusion: the complexity is only linear to the local dynamic changes

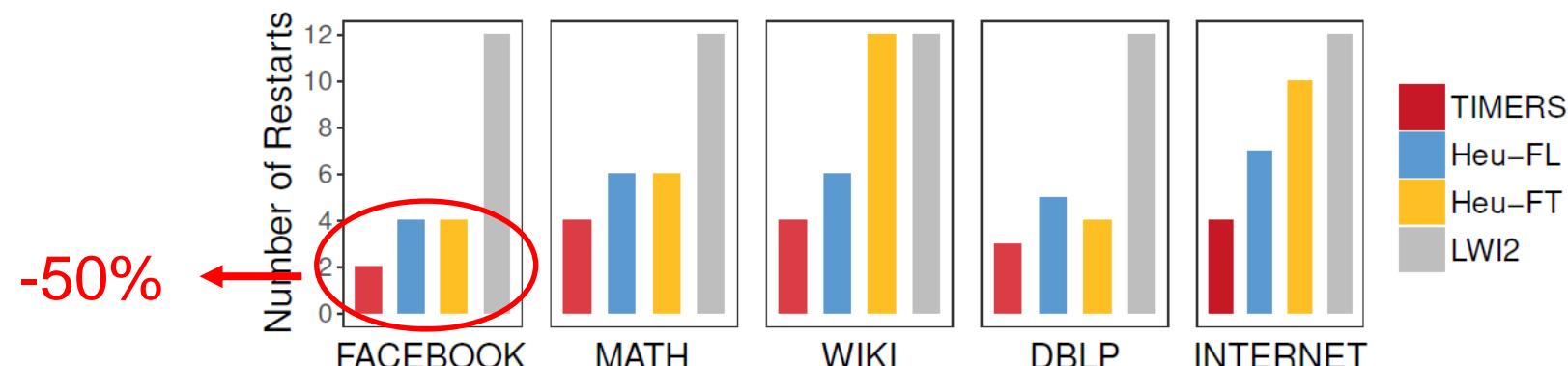
# Experimental Results: Approximation Error

## □ Fixing number of restarts

Dataset	$avg(r)$				$max(r)$			
	TIMERS	LWI2	Heu-FL	Heu-FT	TIMERS	LWI2	Heu-FL	Heu-FT
FACEBOOK	<b>0.005</b>	0.020	0.009	0.011	<b>0.014</b>	0.038	0.025	0.023
MATH	<b>0.037</b>	0.057	0.044	0.051	<b>0.085</b>	0.226	0.117	0.179
WIKI	<b>0.053</b>	0.086	0.071	0.281	<b>0.139</b>	0.332	0.240	0.825
DBLP	<b>0.042</b>	0.110	0.053	0.064	<b>0.121</b>	0.386	0.198	0.238
INTERNET	<b>0.152</b>	0.218	0.196	0.961	<b>0.385</b>	0.806	0.647	1.897

## □ Fixing maximum error

27%~42% Improvement



# Section Summary

- I : Out-of-sample nodes
  - DepthLGP = Non-parametric GP + DNN
- II : Incremental edges
  - DHPE: Generalized Eigen Perturbation
- III: Aggregated error
  - TIMERS: A theoretically guaranteed SVD restart strategy
- IV: Scalable optimization
  - D-SGD: A iteration-wise weighted SGD for highly dynamic data

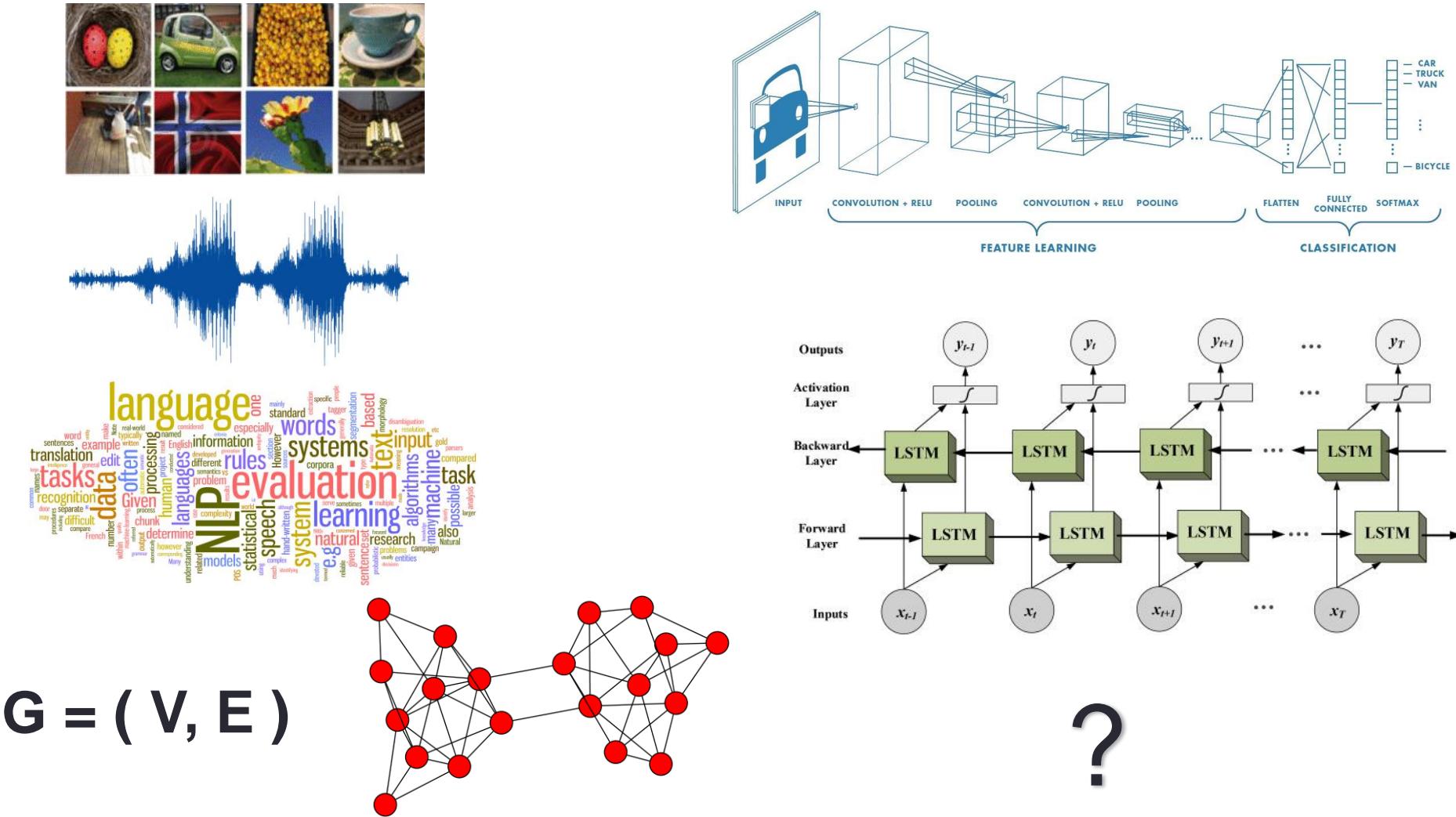
# Recap: Network Embedding

- **Structure-preserved network embedding**
- **Property-preserved network embedding**
- **Dynamic network embedding**

# Learning from networks

GNN

# Graph Neural Networks



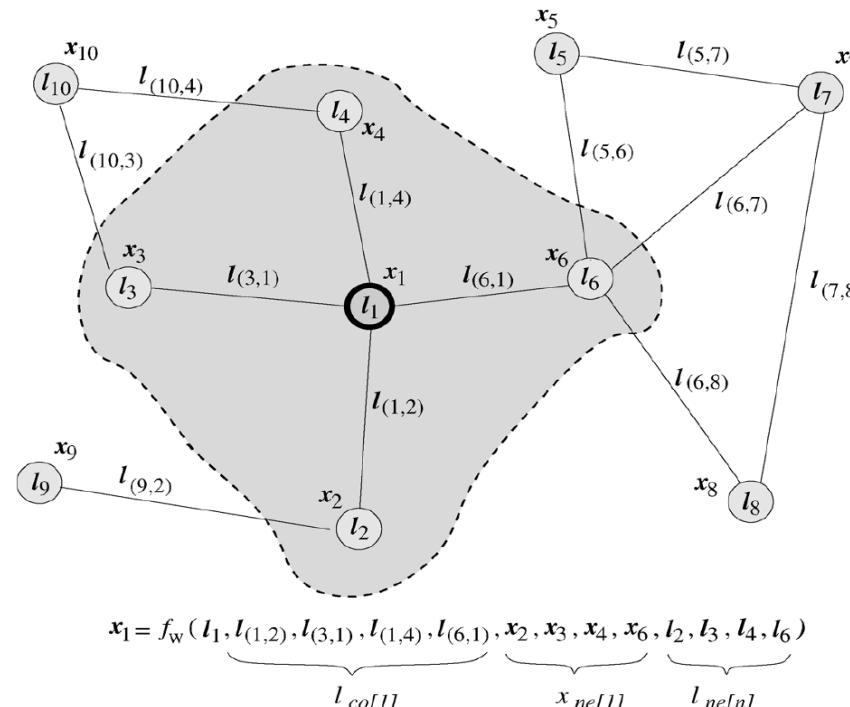
Can we design a learning mechanism to directly work on graphs?

# The First Graph Neural Network

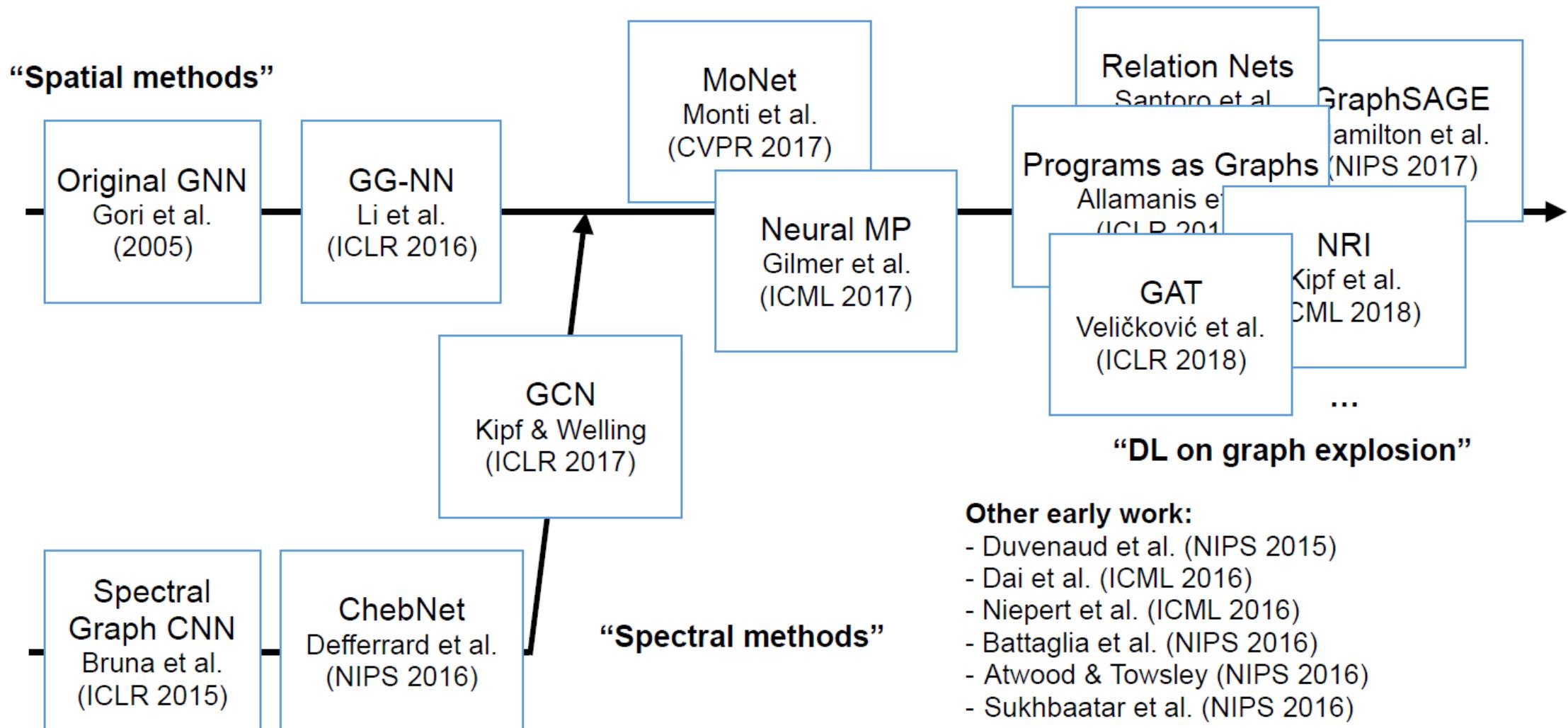
- Basic idea: a recursive definition of states

$$\mathbf{s}_i = \sum_{j \in \mathcal{N}(i)} \mathcal{F} \left( \mathbf{s}_i, \mathbf{s}_j, \mathbf{F}_i^V, \mathbf{F}_j^V, \mathbf{F}_{i,j}^E \right)$$

- A simple example: PageRank

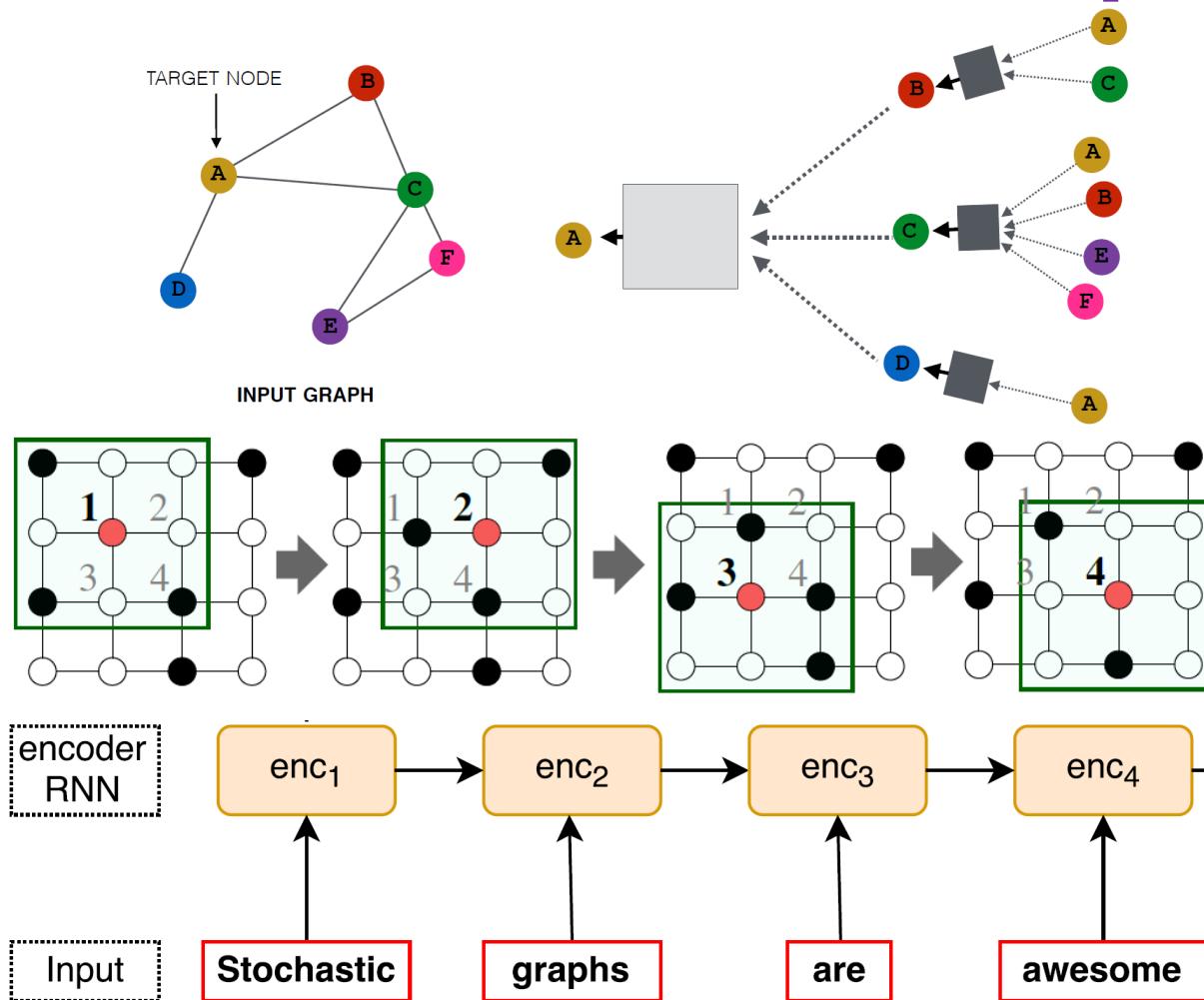


# Many GNNs have emerged since then



(slide inspired by Alexander Gaunt's talk on GNNs)

# How are GNNs compared with other NNs?



- Capture information from graph neighborhoods
- Capture information from nearby grids (i.e., a 2-D graph)
- Capture information from contexts (i.e., a 1-D graph)

We need to exchange information within neighborhoods

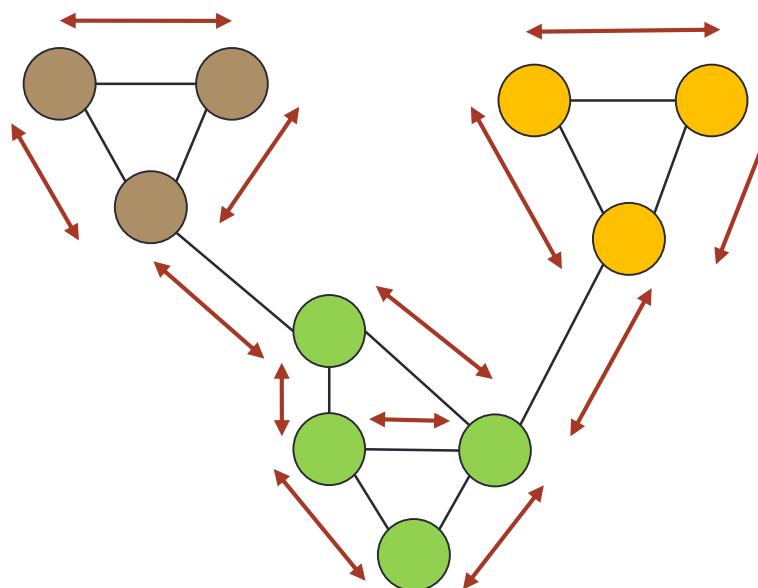
# Message-passing Framework

## Formulation:

$$\mathbf{m}_i^{(l)} = \text{AGG}(\{\mathbf{h}_j^{(l)}, \forall j \in \tilde{\mathcal{N}}_i\})$$

$$\mathbf{h}_i^{(l+1)} = \text{UPDATE}([\mathbf{h}_i^{(l)}, \mathbf{m}_i^{(l)}])$$

- $\mathbf{h}_i^{(l)}$ : representation of node  $v_i$  in the  $l^{th}$  layer
- $\mathbf{m}_i^{(l)}$ : messages for node  $v_i$  in the  $l^{th}$  layer by aggregating neighbor representations



J. Gilmer, et al. Neural message passing for quantum chemistry. *ICML*, 2017.

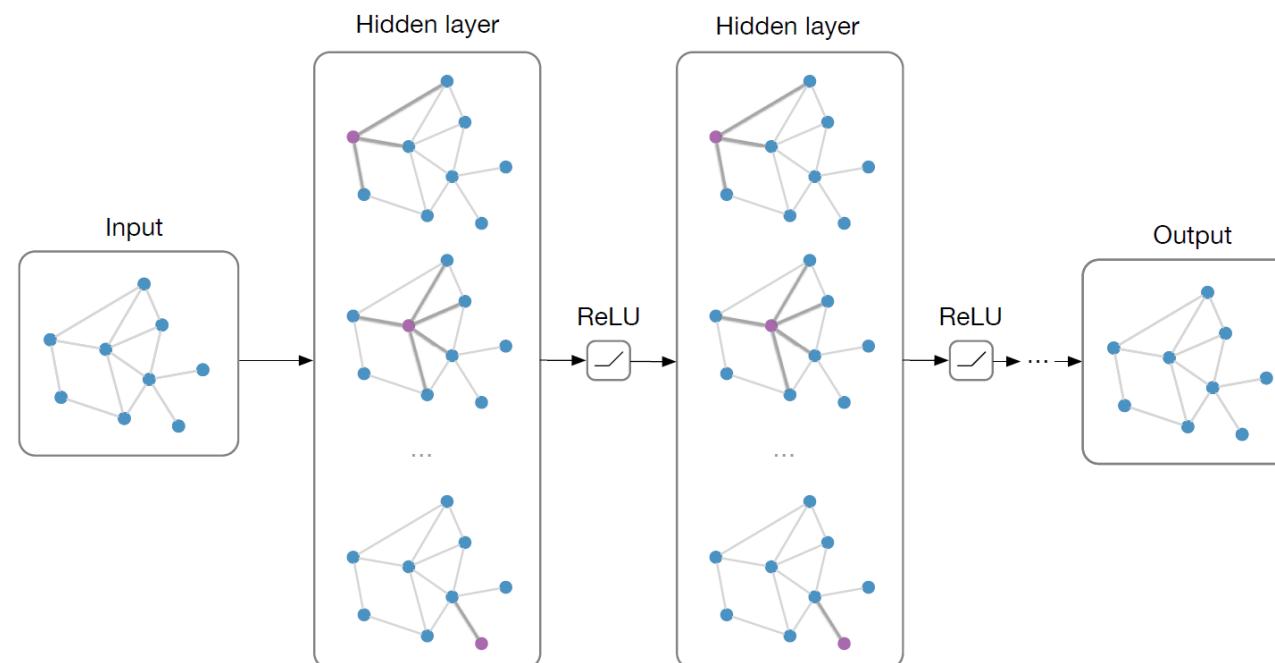
W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *NIPS*, 2017.

# Graph Convolutional Networks (GCN)

- Main idea: averaging messages from direct neighborhoods

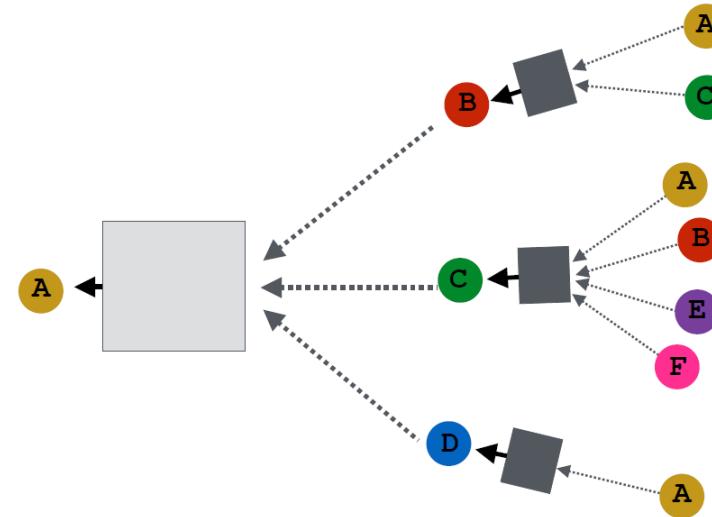
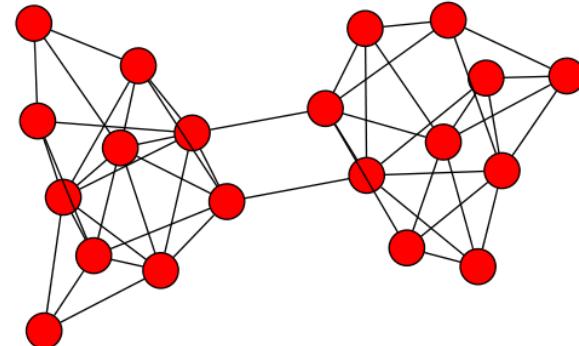
$$\mathbf{H}^{l+1} = \rho \left( \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^l \Theta^l \right)$$

- Stacking multiple layers like standard CNNs:
- State-of-the-art results on node classification



# Expected Properties of GNNs

$$G = (V, E)$$



- Some expected properties of GNNs:
- Trained end-to-end for downstream tasks
  - Vs. network embedding: unsupervised representation learning to handle various tasks
- Utilize node features and graph structures simultaneously
- A deep learning model
- Can handle real applications with data represented as graphs



Are existing  
GNNs good  
enough?

# Outline

- Does GNN fuse *feature* and *topology* optimally?
- Is GNN really a *deep* model?
- Technical challenges in real applications: robustness, explainability and applicability

# Outline

- Does GNN fuse *feature* and *topology* optimally?
- Is GNN really a *deep* model?
- Technical challenges in real applications: robustness, explainability and applicability

# The intrinsic problem GCN is solving

Fusing topology and features in the way of **smoothing features** with the assistance of topology with a **deep** model.

$$\mathbf{H}^{l+1} = \rho \left( \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^l \Theta^l \right)$$

$$\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \times \mathbf{H}^l = \mathbf{D}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^l$$

# Can GNNs Fully Preserve Graph Structures?

- When feature plays the key role, GNN performs good
- How about the contrary?
- Synthesis data: stochastic block model + random features
  - DeepWalk greatly outperforms all the GCNs
  - Recall the message-passing framework

$$\begin{aligned} \mathbf{m}_i^{(l)} &= \text{AGG}(\{\mathbf{h}_j^{(l)}, \forall j \in \tilde{\mathcal{N}}_i\}) \\ \mathbf{h}_i^{(l+1)} &= \text{UPDATE}([\mathbf{h}_i^{(l)}, \mathbf{m}_i^{(l)}]) \end{aligned}$$

Initialized as node features  
Graph structures only provide neighborhoods in aggregation

- Initial node features provide important inductive bias!

Method	Results
Random	10.0
GCN-1	29.7
GCN-2	48.4
GCN-3	56.2
GCN-5	53.3
DeepWalk	99.9

GCN-X: X number of layers

# Can GNNs Fully Preserve Graph Structures?

- Theoretical analysis: node features as “true signal”, GNNs as a low-pass filtering
  - Simplified GCN<sup>[1]</sup>: removing all non-linearity

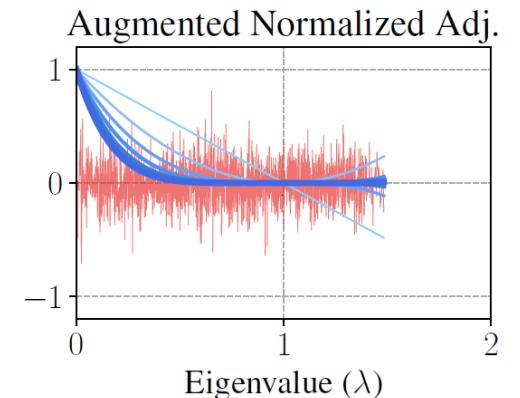
$$H^{(l)} = S^l H^{(0)}$$

$$S = \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} = I - \tilde{L}_{Sym}$$

- From graph signal processing,  $f' = S^l f$  corresponds to a spectral filter <sup>[2]</sup>
- Thus GNNs are inevitably feature-centric
- More recent results on learning graph moments <sup>[3]</sup>:

**Proposition 1.** *A graph convolutional network with  $n$  layers, and no bias terms, in general, can learn  $f(A)_i = \sum_j (A^n)_{ij}$  only if  $n = p$  or  $n > p$  if the bias is allowed.*

- Graph moment: similar to high-order proximities in network embedding



1. Simplifying Graph Convolutional Networks, *ICML 2019*
2. Revisiting Graph Neural Networks All We Have is Low-Pass Filters, *arXiv 1905.09550*
3. Understanding the Representation Power of Graph Neural Networks in Learning Graph Topology, *NeurIPS 2019*

# A New Perspective to Understand GNNs

- How can we empower GNNs to preserve graph structures well?
- A new perspective: treating GNNs as a type of (non-linear) dimensionality reduction

- A slightly modified framework:

$$\mathbf{H}^{(l+1)} = \sigma \left( \mathcal{F}(\mathbf{A}) \mathbf{H}^{(l)} \mathbf{W}^{(l)} \right)$$

Non-linear mapping      Graph structures      Previous bases      Linear transform

Method	$\mathcal{F}(\mathbf{A})$
GCN, SGC	$\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}}$
DCNN	$(\mathbf{D}^{-1} \mathbf{A})^K$
DGCN	$\mathbf{D}_P^{-\frac{1}{2}} \mathbf{A}_P \mathbf{D}_P^{-\frac{1}{2}}$
PPNP	$\alpha \left( \mathbf{I}_N - (1 - \alpha) \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \right)^{-1}$

- Three-steps

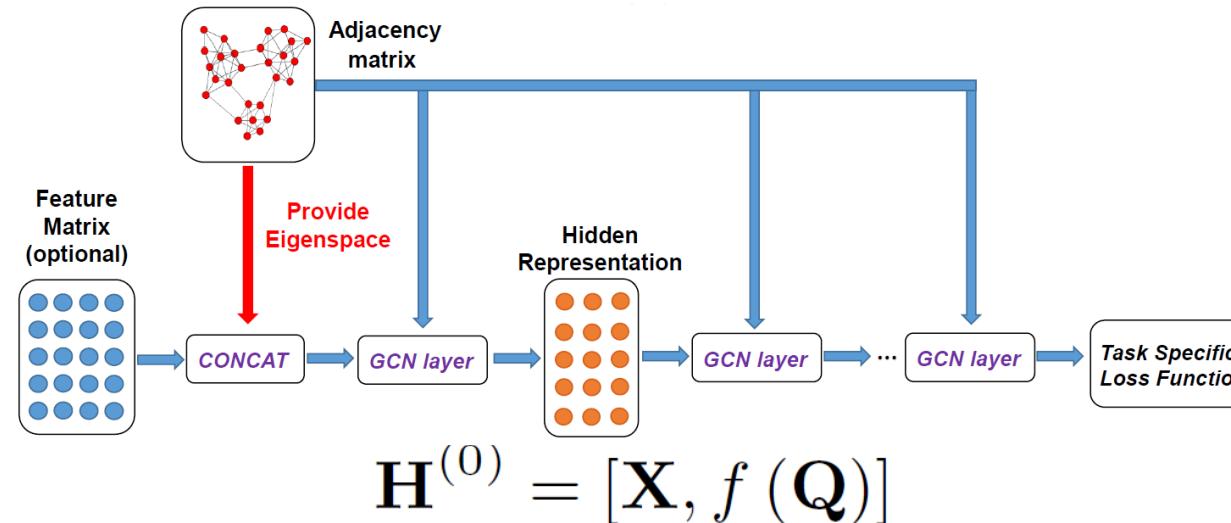
- (1) Projecting graph structures into a subspace spanned by node representations in the last step
- (2) The projected representations are linearly transformed followed by a non-linear mapping
- (3) Repeat the process by using the new node representations as bases

- Why are the existing GNNs feature-centric?

→ The initial space is solely determined by node features!

# Eigen-GNN: A Graph Structure Preserving Plug-in

## □ Framework



X: node features; Q: top-d eigenvector of a graph structure matrix;  $f(\cdot)$ : a simple function such as normalization

## □ Experimental results:

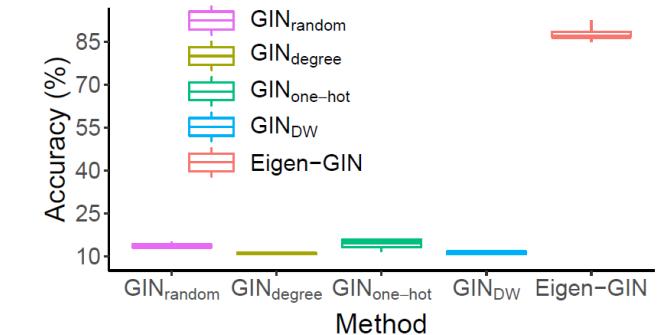
### Node classification

Data	Method	Harvard	Columbia	Stanford
A, Y	GCN <sub>random</sub>	74.6 ± 0.5	63.6 ± 1.6	68.2 ± 1.5
	GCN <sub>degree</sub>	74.4 ± 2.0	63.8 ± 2.3	67.8 ± 1.6
	GCN <sub>DW</sub>	82.5 ± 1.0	<b>76.0 ± 1.3</b>	76.6 ± 1.3
	Eigen-GCN <sub>struc</sub>	<b>82.7 ± 1.2</b>	<b>76.0 ± 1.9</b>	<b>78.9 ± 1.3</b>
A, X, Y	GCN <sub>feat</sub>	70.6 ± 1.3	74.8 ± 1.7	71.3 ± 1.6
	GCN <sub>feat+DW</sub>	83.1 ± 0.7	77.6 ± 1.3	78.3 ± 1.4
	Eigen-GCN <sub>feat+struc</sub>	<b>84.6 ± 1.4</b>	<b>78.6 ± 1.1</b>	<b>79.7 ± 1.2</b>

### Link Prediction

Dataset	C.elegans	E.coli	Power
SEAL	77.6 ± 0.9	91.5 ± 0.8	69.9 ± 1.6
SEAL <sub>DW</sub>	77.1 ± 1.5	92.1 ± 0.7	69.8 ± 1.5
Eigen-SEAL	<b>79.5 ± 0.8*</b>	<b>92.5 ± 0.6*</b>	<b>73.2 ± 2.4*</b>
Gain†	+1.9	+0.4	+3.3

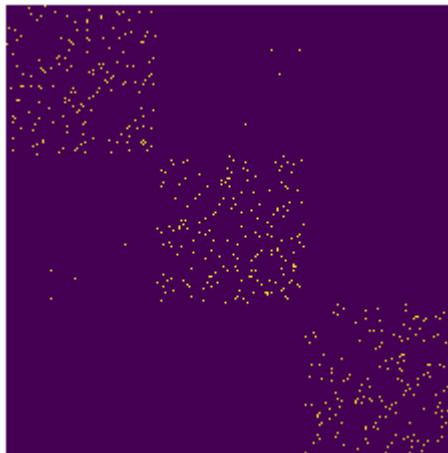
### Graph Isomorphism Test



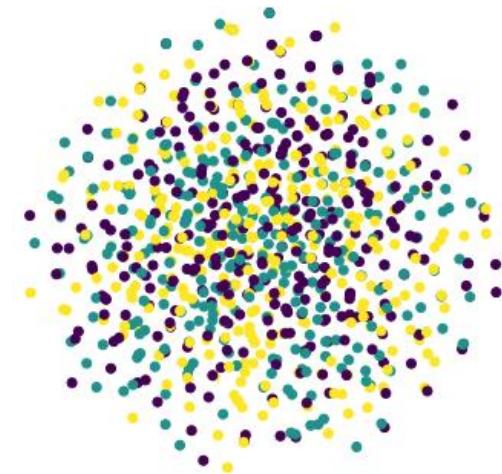
# Can GNNs Fully Preserve Node Features?

- Though GNNs have a feature-driven mechanism, can they preserve node features well?

**Case 1**



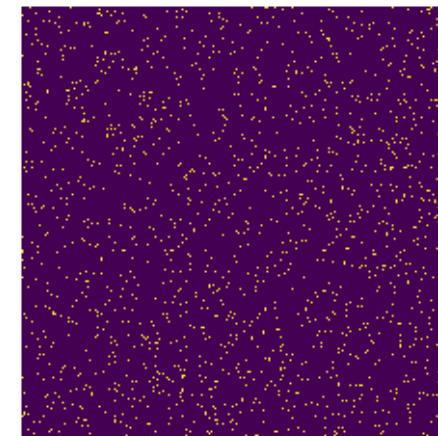
Correlated Topology



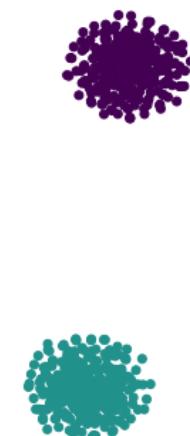
Random Features

DeepWalk > GCN

**Case 2**



Random topology



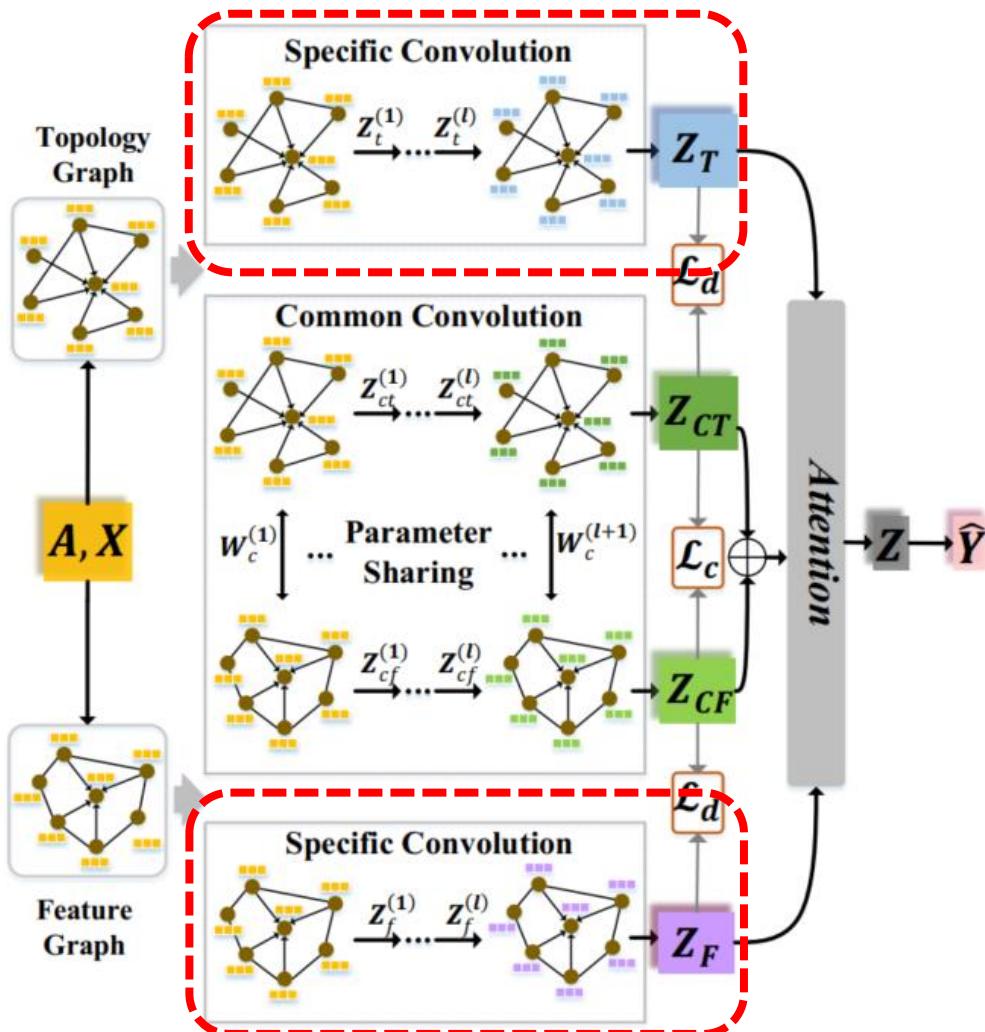
Correlated Features

MLP > GCN

- Reason: the neighborhood only depends on structures (i.e., not depend on features)
- How to find the most suitable neighborhoods adaptively?

**GNNs also fail in preserving node features!**

# Adaptive Multi-channel GCN



## Specific Convolution Module

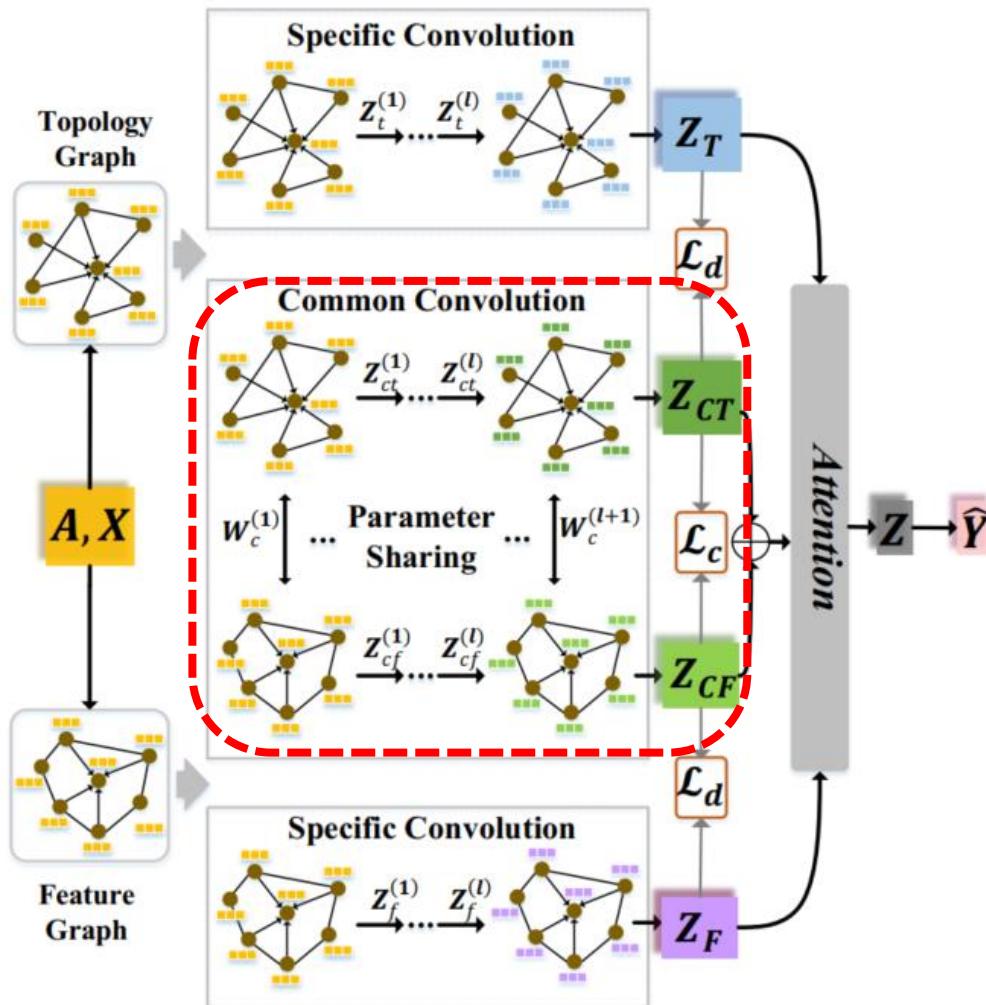
□ **Topology Graph**  $G_t = (A, X)$

$$Z_t^{(l)} = \text{ReLU}(\tilde{\mathbf{D}}_t^{-\frac{1}{2}} \tilde{\mathbf{A}}_t \tilde{\mathbf{D}}_t^{-\frac{1}{2}} Z_t^{(l-1)} \mathbf{W}_t^{(l)}),$$

□ **Feature Graph**  $G_f = (A_f, X_f)$

$$Z_f^{(l)} = \text{ReLU}(\tilde{\mathbf{D}}_f^{-\frac{1}{2}} \tilde{\mathbf{A}}_f \tilde{\mathbf{D}}_f^{-\frac{1}{2}} Z_f^{(l-1)} \mathbf{W}_f^{(l)}),$$

# Adaptive Multi-channel GCN



## Common Convolution Module

□ **Topology Graph**  $G_t = (A, X)$

$$Z_{ct}^{(l)} = \text{ReLU}(\tilde{\mathbf{D}}_t^{-\frac{1}{2}} \tilde{\mathbf{A}}_t \tilde{\mathbf{D}}_t^{-\frac{1}{2}} Z_{ct}^{(l-1)} \mathbf{W}_c^{(l)}),$$

□ **Feature Graph**  $G_f = (A_f, X)$

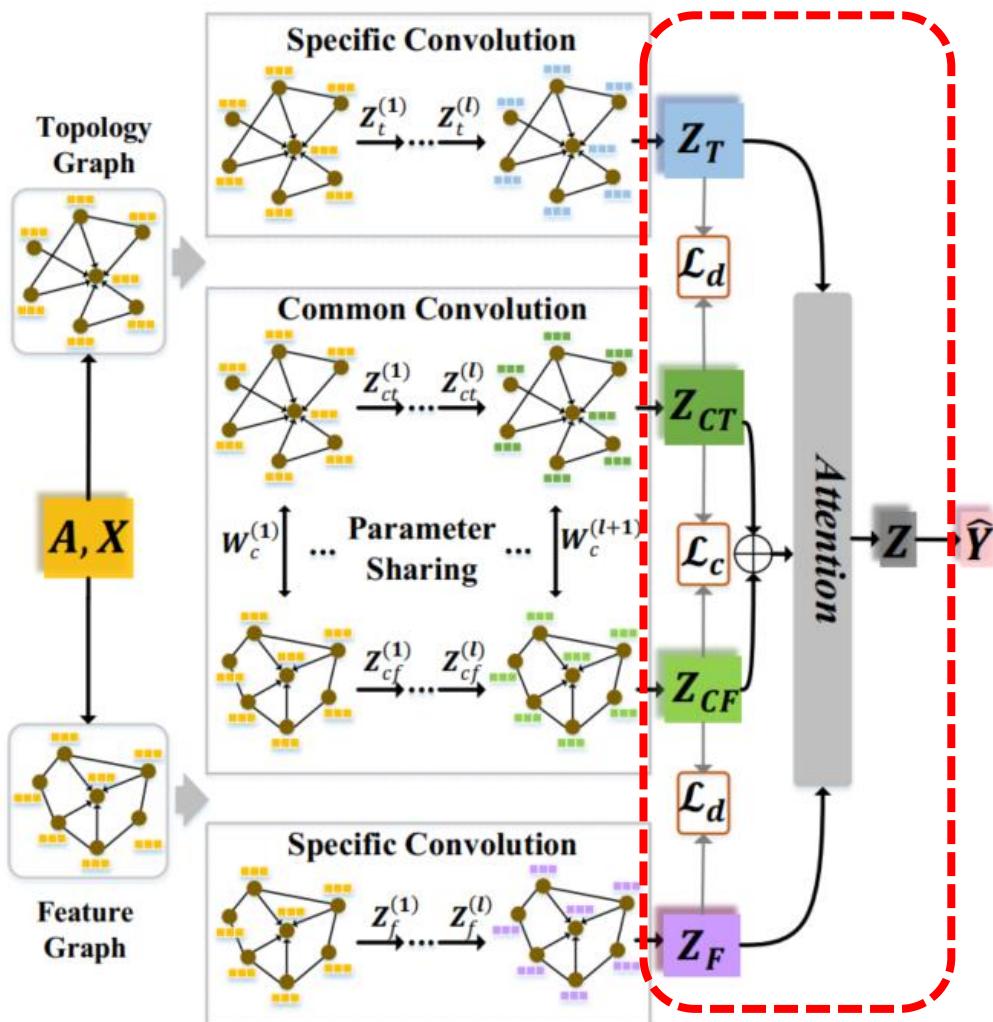
$$Z_{cf}^{(l)} = \text{ReLU}(\tilde{\mathbf{D}}_f^{-\frac{1}{2}} \tilde{\mathbf{A}}_f \tilde{\mathbf{D}}_f^{-\frac{1}{2}} Z_{cf}^{(l-1)} \mathbf{W}_c^{(l)}),$$

□ **Parameter Sharing**

□ **Common Embedding**

$$Z_C = (Z_{CT} + Z_{CF})/2.$$

# Adaptive Multi-channel GCN



## Attention Mechanism

$$(\alpha_t, \alpha_c, \alpha_f) = att(Z_T, Z_C, Z_F),$$

$$\alpha_T = diag(\alpha_t) \quad \alpha_C = diag(\alpha_c) \quad \alpha_F = diag(\alpha_f).$$

$$Z = \alpha_T \cdot Z_T + \alpha_C \cdot Z_C + \alpha_F \cdot Z_F.$$

### □ How to Calculate $\alpha_t$

$$\omega_T^i = \mathbf{q}^T \cdot \tanh(\mathbf{W}_T \cdot (z_T^i)^T + \mathbf{b}_T).$$

$$\alpha_T^i = softmax(\omega_T^i) = \frac{\exp(\omega_T^i)}{\exp(\omega_T^i) + \exp(\omega_C^i) + \exp(\omega_F^i)}.$$

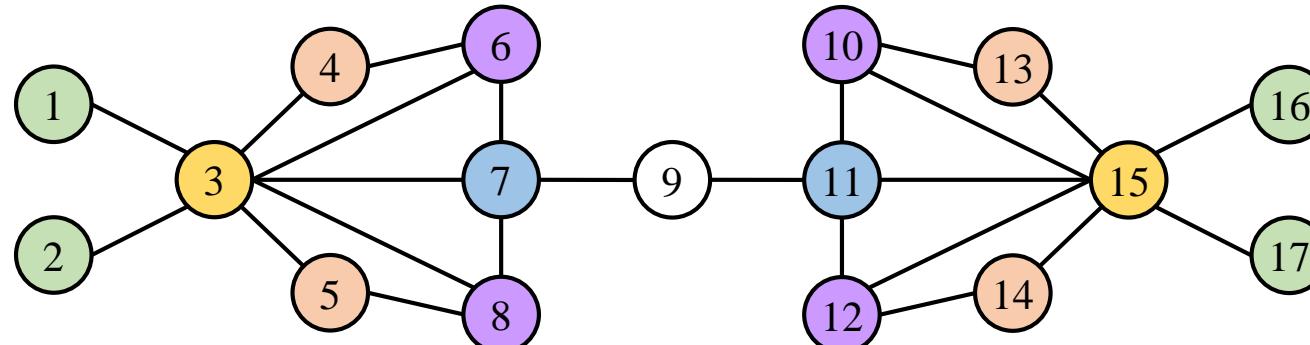
$$\alpha_t = [\alpha_T^i]$$

# Experimental Results

Datasets	Metrics	L/C	DeepWalk	LINE	Chebyshev	GCN	kNN-GCN	GAT	DEMO-Net	MixHop	AM-GCN
Citeseer	ACC	20	43.47	32.71	69.80	70.30	61.35	<u>72.50</u>	69.50	71.40	<b>73.30</b>
		40	45.15	33.32	71.64	<u>73.10</u>	61.54	73.04	70.44	71.48	<b>74.70</b>
		60	48.86	35.39	73.26	74.48	62.38	<u>74.76</u>	71.86	72.16	<b>75.56</b>
	F1	20	38.09	31.75	65.92	67.50	58.86	<u>68.14</u>	67.84	66.96	<b>69.22</b>
		40	43.18	32.42	68.31	<u>69.70</u>	59.33	69.58	66.97	67.40	<b>69.81</b>
		60	48.01	34.37	70.31	<u>71.24</u>	60.07	<b>71.60</b>	68.22	69.31	70.92
UAI2010	ACC	20	42.02	43.47	50.02	49.88	<u>66.06</u>	56.92	23.45	61.56	<b>70.46</b>
		40	51.26	45.37	58.18	51.80	<u>68.74</u>	63.74	30.29	65.05	<b>73.14</b>
		60	54.37	51.05	59.82	54.40	<u>71.64</u>	68.44	34.11	67.66	<b>74.40</b>
	F1	20	32.93	37.01	33.65	32.86	<u>52.43</u>	39.61	16.82	49.19	<b>55.61</b>
		40	46.01	39.62	38.80	33.80	<u>54.45</u>	45.08	26.36	53.86	<b>64.88</b>
		60	44.43	43.76	40.60	34.12	54.78	48.97	29.05	<u>56.31</u>	<b>65.99</b>
ACM	ACC	20	62.69	41.28	75.24	<u>87.80</u>	78.52	87.36	84.48	81.08	<b>90.70</b>
		40	63.00	45.83	81.64	<u>89.06</u>	81.66	88.60	85.70	82.34	<b>90.76</b>
		60	67.03	50.41	85.43	<u>90.54</u>	82.00	90.40	86.55	83.09	<b>91.42</b>
	F1	20	62.11	40.12	74.86	<u>87.82</u>	78.14	87.44	84.16	81.40	<b>90.63</b>
		40	61.88	45.79	81.26	<u>89.00</u>	81.53	88.55	84.83	81.13	<b>90.66</b>
		60	66.99	49.92	85.26	<u>90.49</u>	81.95	90.39	84.05	82.24	<b>91.36</b>

# Permutation-equivariance of GNNs

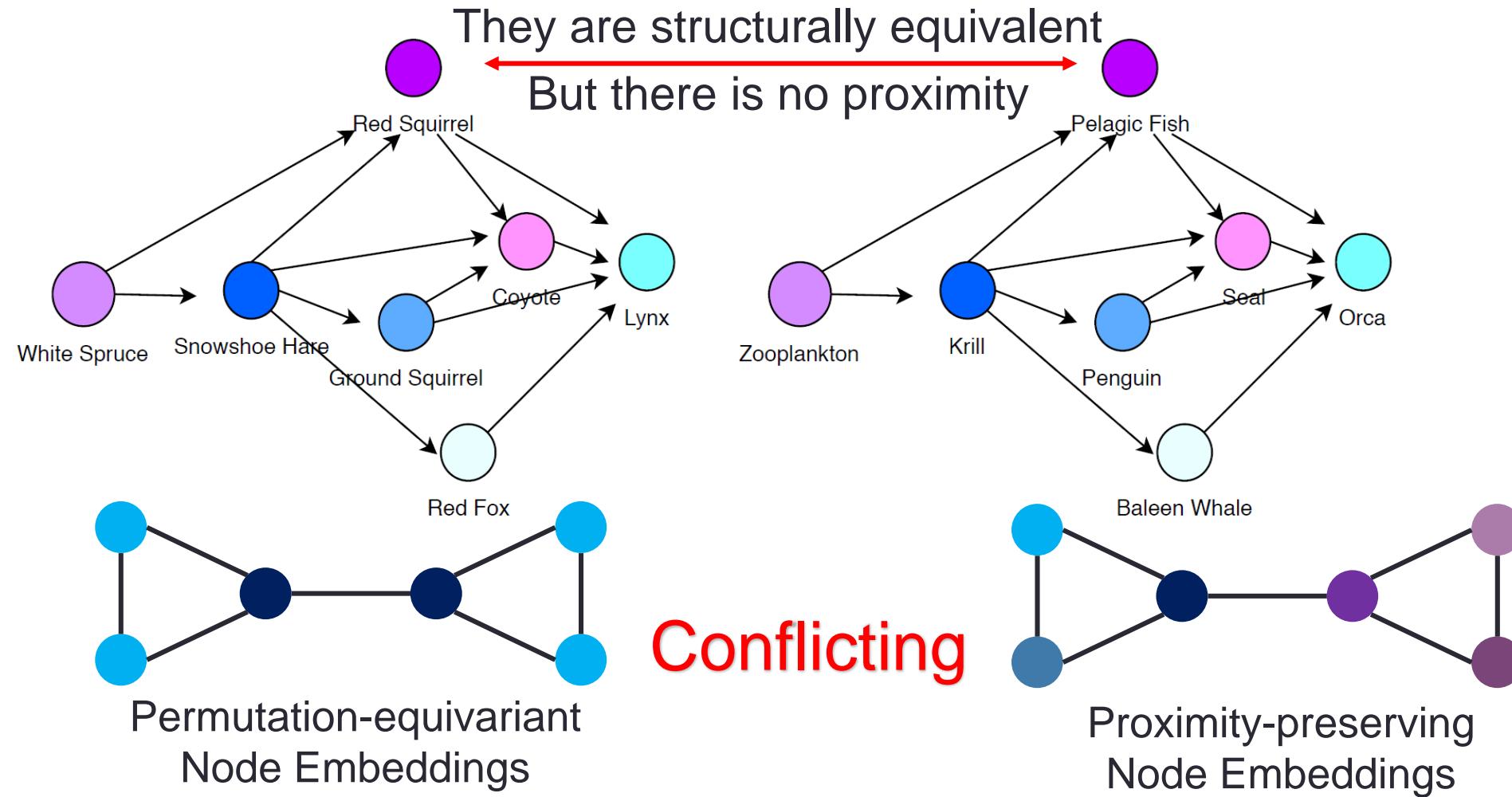
- Permutation-equivariance property
  - If we randomly permute the IDs of nodes while maintaining the graph structure, the representations of nodes in GNNs should be permuted accordingly
- Pros:
  - Guarantees that the embeddings of automorphic nodes are identical
  - Automatically generalize to all the  $O(N!)$  permutations when training with only one permutation
- All the existing message-passing GNNs satisfy permutation-equivariance



Permutation-equivariant Node Embeddings

# Permutation-equivariance vs. Proximity-aware

- However, permutation-equivariance and proximity-aware are conflicting

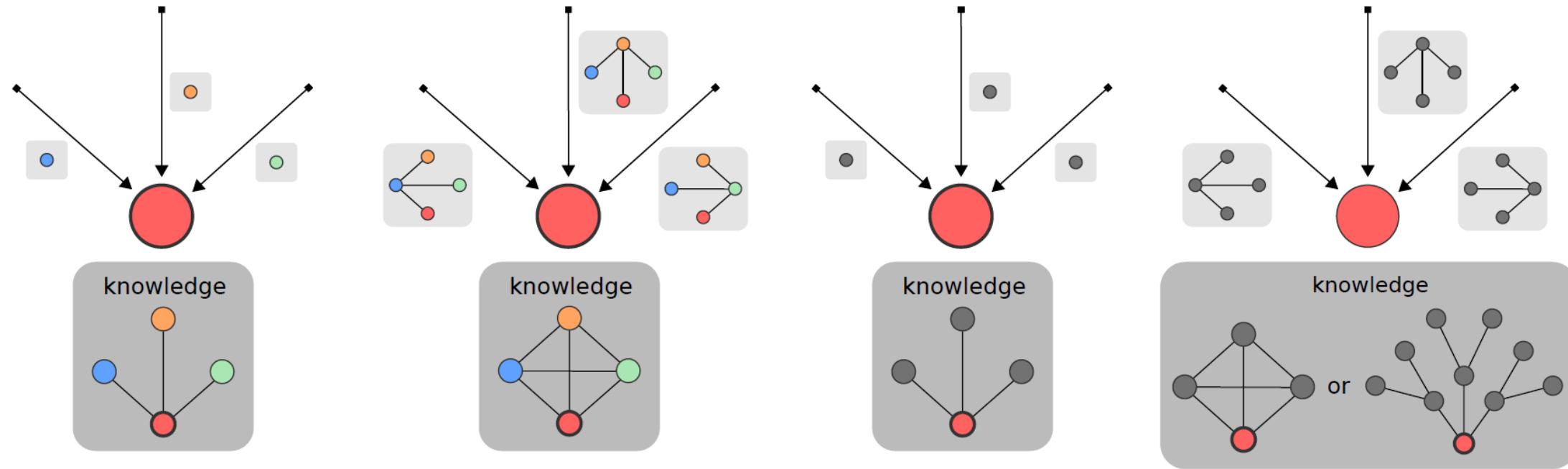


Position-aware graph neural networks. *ICML* 2019.

On the Equivalence between Positional Node Embeddings and Structural Graph Representations. *ICLR* 2020.

# Unique Node Identifiers

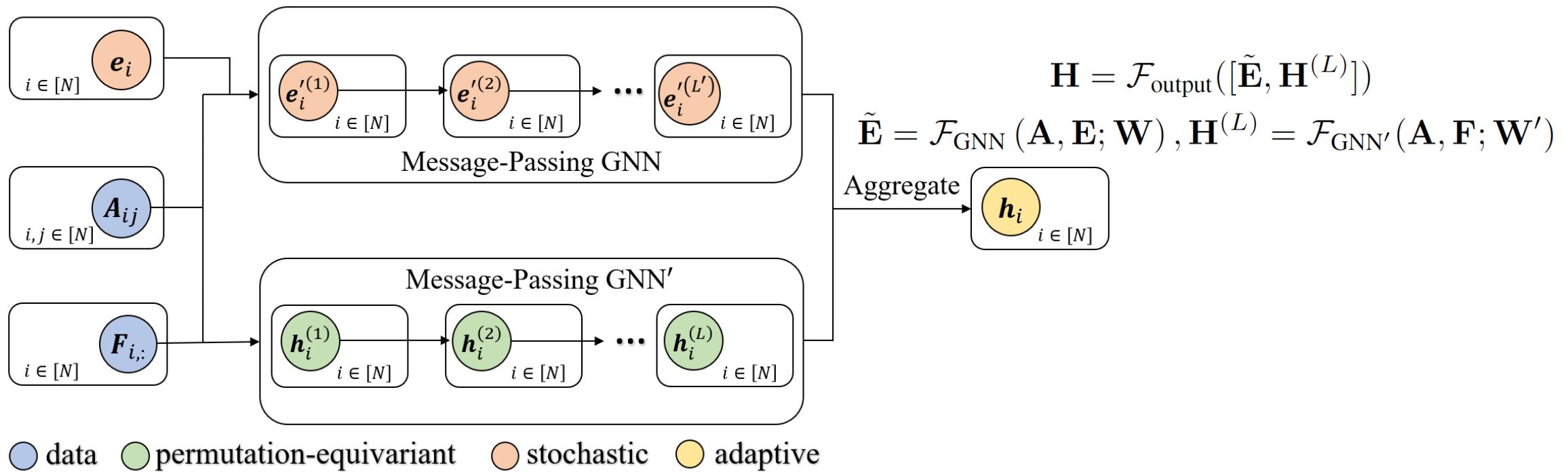
- The key problem is that nodes cannot differentiate each other



- Theoretical analysis: unique node identifiers are one necessary condition for GNNs to be universal approximation

# Stochastic Message Passing (SMP)

- Assign stochastic features as node identifiers
  - Gaussian features: associate with random projection literature
- A dual GNN architecture:



# Theoretical Guarantee

- SMP can preserve node proximities

**Theorem 2.** An SMP in Eq. (9) with the message-passing matrix  $\tilde{\mathbf{A}}$  and the number of propagation steps  $K$  can preserve the walk-based proximity  $\tilde{\mathbf{A}}^K (\tilde{\mathbf{A}}^K)^T$  with high probability if the dimensionality of the stochastic matrix  $d$  is sufficiently large, where the superscript  $T$  denotes matrix transpose. The theorem is regardless of whether  $\mathbf{E}$  are fixed or resampled.

- SMP can recover the existing permutation-equivariant GNNs

**Corollary 2.** For any task, Eq. (8) with the aforementioned linear  $\mathcal{F}_{\text{output}}(\cdot)$  is at least as powerful as the permutation-equivariant  $\mathcal{F}_{\text{GNN}'}(\mathbf{A}, \mathbf{F}; \mathbf{W}')$ , i.e., the minimum training loss of using  $\mathbf{H}$  in Eq. (8) is equal to or smaller than using  $\mathbf{H}^{(L)} = \mathcal{F}_{\text{GNN}'}(\mathbf{A}, \mathbf{F}; \mathbf{W}')$ .

- An adaptive GNN that maintains both proximity-awareness and permutation-equivariance

# Experimental Results

Table 2: The results of link prediction tasks measured in AUC (%). The best results and the second-best results for each dataset, respectively, are in bold and underlined.

Model	Grid	Communities	Email	CS	Physics	PPI
SGC	57.6±3.8	51.9±1.6	68.5±7.0	96.5±0.1	<b>96.6±0.1</b>	80.5±0.4
GCN	61.8±3.6	50.3±2.5	67.4±6.9	93.4±0.3	93.8±0.2	78.0±0.4
GAT	61.0±5.5	51.1±1.6	53.5±6.3	93.7±0.9	94.1±0.4	79.3±0.5
PGNN <sup>5</sup>	<u>73.4±6.0</u>	97.8±0.6	70.9±6.4	82.2±0.5	Out of memory	80.8±0.4
SMP-Identity	55.1±4.8	<b>98.0±0.7</b>	<u>72.9±5.1</u>	<u>96.5±0.1</u>	<u>96.5±0.1</u>	<u>81.0±0.2</u>
SMP-Linear	<b>73.6±6.2</b>	97.7±0.5	<b>75.7±5.0</b>	<b>96.7±0.1</b>	96.1±0.1	<b>81.9±0.3</b>

Superior performance when the task is proximity-related

Table 3: The results of node classification tasks measured by accuracy (%). The best results and the second-best results for each dataset, respectively, are in bold and underlined.

Model	Communities	CS	Physics	Cora	CiteSeer	PubMed
SGC	7.1±2.1	67.2±12.8	92.3±1.6	76.9±0.2	63.6±0.0	74.2±0.1
GCN	<u>7.5±1.2</u>	<u>91.1±0.7</u>	<u>93.1±0.8</u>	<u>81.4±0.5</u>	<b>71.3±0.5</b>	<b>79.3±0.4</b>
GAT	5.0±0.0	90.5±0.5	<b>93.1±0.4</b>	<b>82.9±0.5</b>	<u>71.2±0.6</u>	<u>77.9±0.5</u>
PGNN	5.2±0.5	77.6±7.6	Out of memory	59.2±1.5	55.7±0.9	Out of memory
SMP-Linear	<b>99.9±0.3</b>	<b>91.5±0.8</b>	<u>93.1±0.8</u>	80.9±0.8	68.2±1.0	76.5±0.8

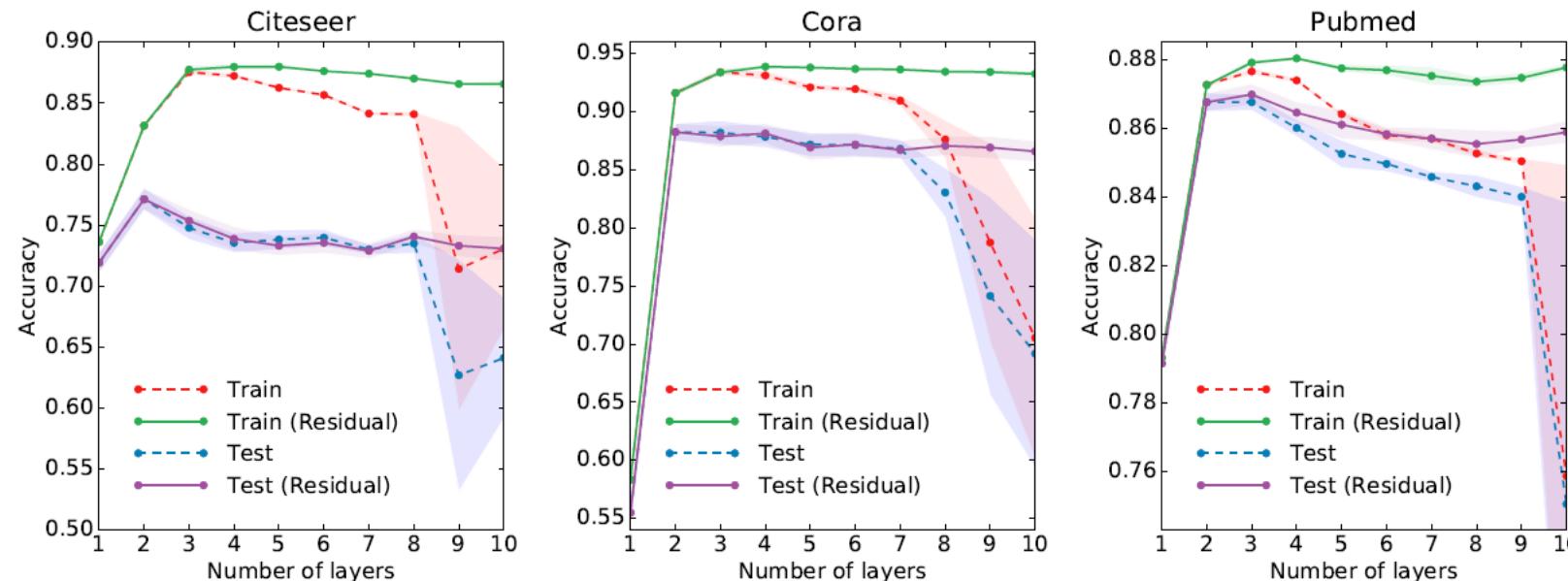
Comparable performance when permutation-equivariant is helpful

# Outline

- Does GNN fuse *feature* and *topology* optimally?
- Is GNN really a *deep* model?
- Technical challenges in real applications: robustness, explainability and applicability

# Is GNN really a deep model?

- Though GNNs are motivated as “deep learning”, most models only adopt 2-3 layers



- Possible reasons:
  - Difficulties in training deep networks, e.g., over-fitting, vanishing gradients
  - Over-smoothing, i.e. all nodes have similar representations in deeper layers

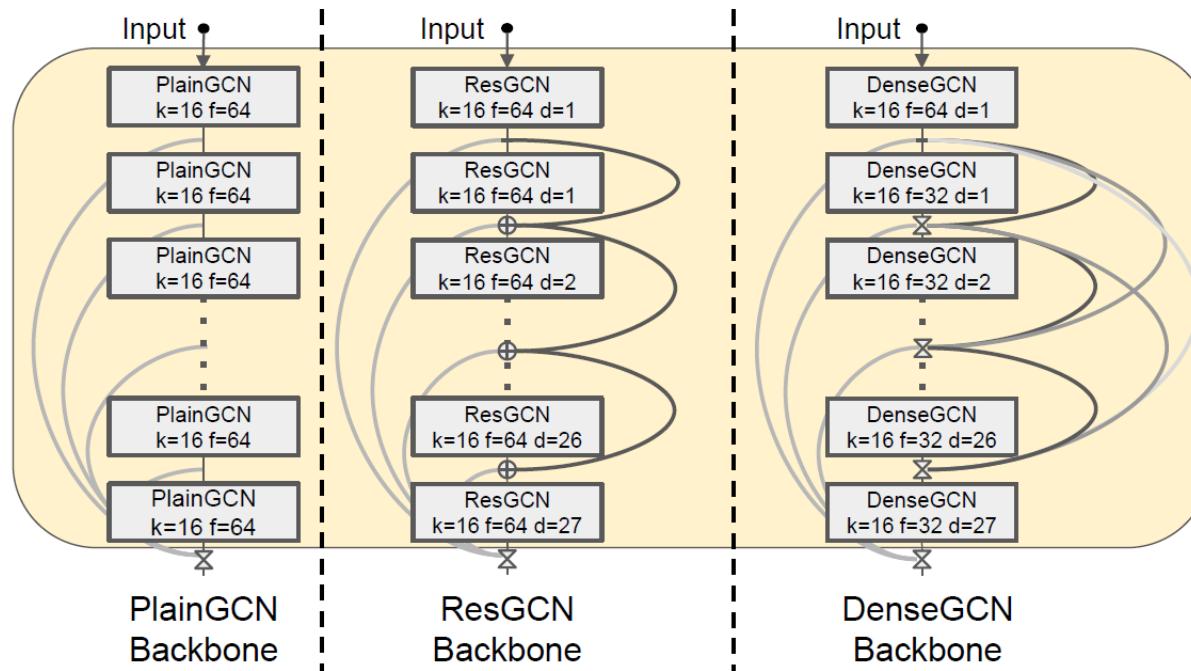
Semi-Supervised Classification with Graph Convolutional Networks, *ICLR* 2017.

Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning, *AAAI* 2018.

Graph Neural Networks Exponentially Lose Expressive Power for Node Classification, *ICLR* 2020.

# Is GNN really a deep model?

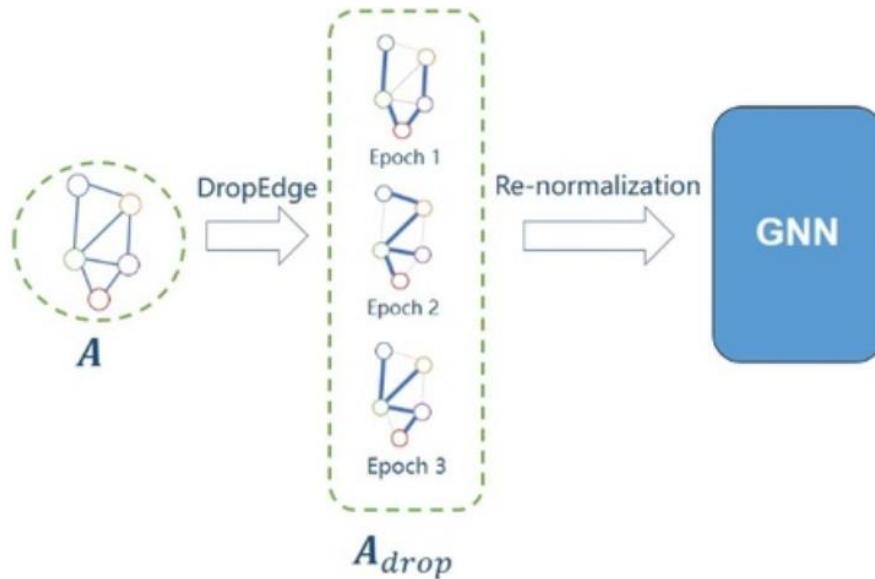
- State-of-the-art CNNs can have hundreds of layers, e.g., ResNet, DenseNet
  - Can we mimic them and use similar ideas to develop GNNs?
- Residual connections, dense connections, and dilated aggregation



Method	OA	mIOU
PointNet [27]	78.5	47.6
MS+CU [8]	79.2	47.8
G+RCU [8]	81.1	49.7
PointNet++ [29]	-	53.2
3DRNN+CF [48]	<b>86.9</b>	56.3
DGCNN [42]	84.1	56.1
<b>ResGCN-28 (Ours)</b>	85.9	<b>60.0</b>

# More Recent Approaches

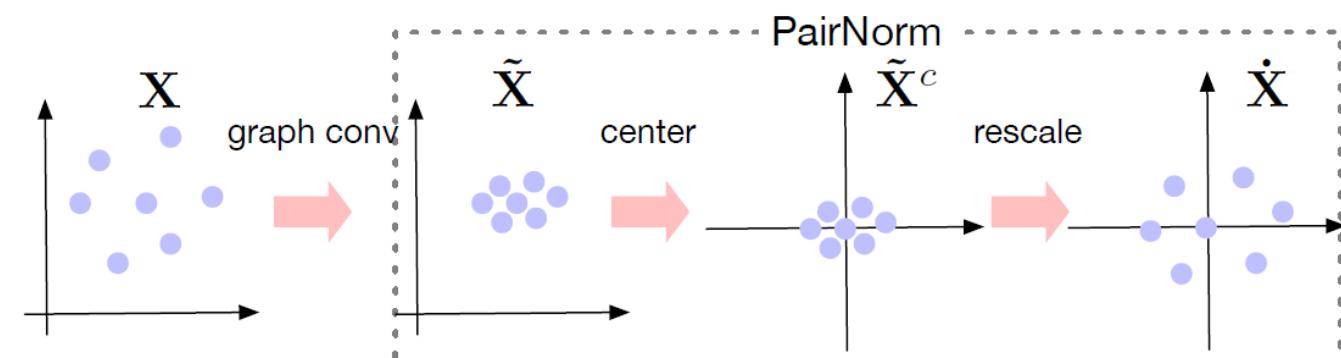
- Design specific dropout to graph data  
→ Randomly drop edges in each epoch



- Theoretical insights:
  - A data augmentation technique
  - A message passing reducer

DropEdge: Towards Deep Graph Convolutional Networks on Node Classification, *ICLR 2020*

- How to prevent node embedding from being similar?  
→ Make total pairwise feature distances remained a constant across layers



$$\tilde{x}_i^c = \tilde{x}_i - \frac{1}{n} \sum_{i=1}^n \tilde{x}_i \quad (\text{Center})$$

$$\dot{x}_i = s \cdot \frac{\tilde{x}_i^c}{\sqrt{\frac{1}{n} \sum_{i=1}^n \|\tilde{x}_i^c\|_F^2}} = s\sqrt{n} \cdot \frac{\tilde{x}_i^c}{\sqrt{\|\tilde{X}^c\|_F^2}} \quad (\text{Scale})$$

PairNorm: Tackling Oversmoothing in GNNs, *ICLR 2020*

# A Hot Research Topic

- How to train real deep GNNs is still an on-going research topic
  - DeepGCNs, DropEdge, PairNorm
  - Many more:
    - Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View, *AAAI 2020*
    - Towards Deeper Graph Neural Networks, *KDD 2020*
    - What graph neural networks cannot learn: depth vs width, *ICLR 2020*
    - Simple and deep GNN, *ICML 2020*
    - Bayesian Graph Neural Networks with Adaptive Connection Sampling, *ICML 2020*
    - An Anatomy of Graph Neural Networks Going Deep via the Lens of Mutual Information: Exponential Decay vs. Full Preservation, *arXiv 1910.04499*
    - Revisiting Oversmoothing in Deep GCNs, *arXiv 2003.13663*
    - Towards Deeper Graph Neural Networks with Differentiable Group Normalization, *arXiv 2006.06972*
    - DeeperGCN: All You Need to Train Deeper GCNs, *arXiv 2006.07739*
    - Effective Training Strategies for Deep Graph Neural Networks, *arXiv 2006.07107*
    - Graphs, Entities, and Step Mixture, *arXiv 2005.08485*

# Cautious: whether and when GNNs need to go deep?

□ Cautious: whether and when do we need deep GNNs?

- A wider neighborhood structure? The extra non-linearity?

□ Recall Simplified GCN<sup>[1]</sup> formulation:

$$H^{(k+1)} = SH^{(k)}W^{(k)},$$

$$\hat{Y} = \text{softmax}(S^K H^{(0)} W)$$
High-order proximity

□ There are also recent theoretical results<sup>[2]</sup>:

<i>problem</i>	<i>bound</i>	<i>problem</i>	<i>bound</i>
cycle detection (odd)	$dw = \Omega(n/\log n)$	shortest path	$d\sqrt{w} = \Omega(\sqrt{n}/\log n)$
cycle detection (even)	$dw = \Omega(\sqrt{n}/\log n)$	max. indep. set	$dw = \Omega(n^2/\log^2 n)$ for $w = O(1)$
subgraph verification*	$d\sqrt{w} = \Omega(\sqrt{n}/\log n)$	min. vertex cover	$dw = \Omega(n^2/\log^2 n)$ for $w = O(1)$
min. spanning tree	$d\sqrt{w} = \Omega(\sqrt{n}/\log n)$	perfect coloring	$dw = \Omega(n^2/\log^2 n)$ for $w = O(1)$
min. cut	$d\sqrt{w} = \Omega(\sqrt{n}/\log n)$	girth 2-approx.	$dw = \Omega(\sqrt{n}/\log n)$
diam. computation	$dw = \Omega(n/\log n)$	diam. $\frac{3}{2}$ -approx.	$dw = \Omega(\sqrt{n}/\log n)$

# Outline

- Does GNN fuse *feature* and *topology* optimally?
- Is GNN really a *deep* model?
- Technical challenges in real applications: robustness,  
explainability and applicability

# Technical challenges in real applications

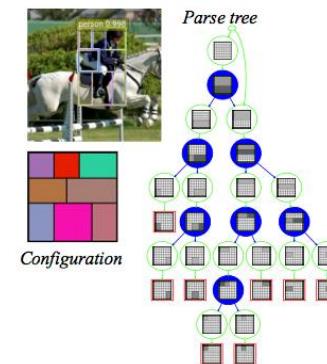


Hot directions in computer vision:

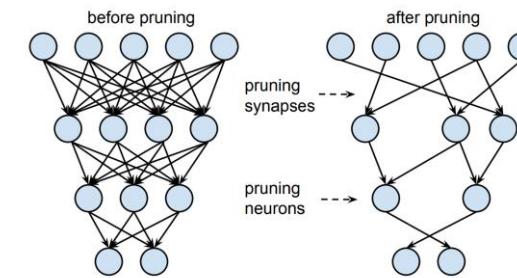
Adversarial



Explainable



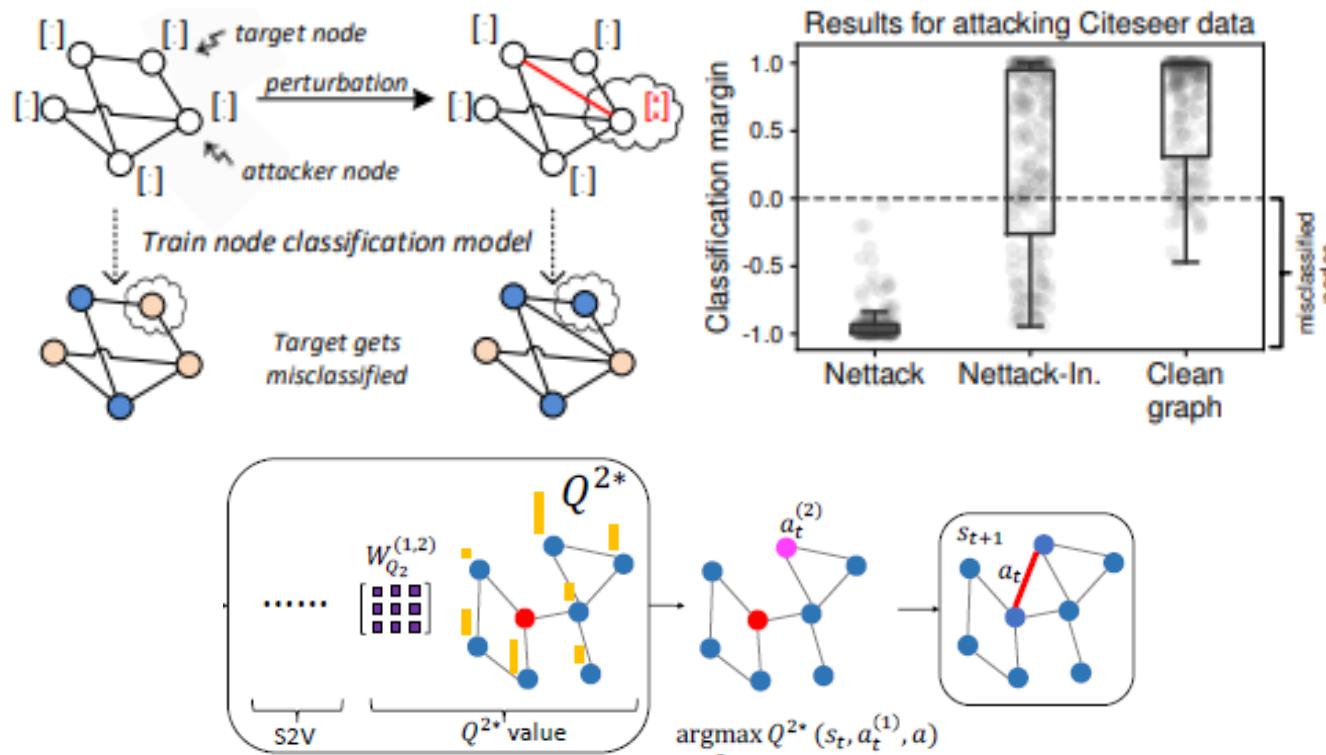
Scalable



# Robustness in GNNs

## □ Adversarial attacks

- Small perturbations in graph structures and node attributes
- Great challenges for applying GNNs to node classification



# Adversarial Attacks on GNNs

## □ Categories

### □ Targeted vs. Non-targeted

- Targeted: the attacker focus on misclassifying some target nodes
- Non-targeted: the attacker aims to reduce the overall model performance

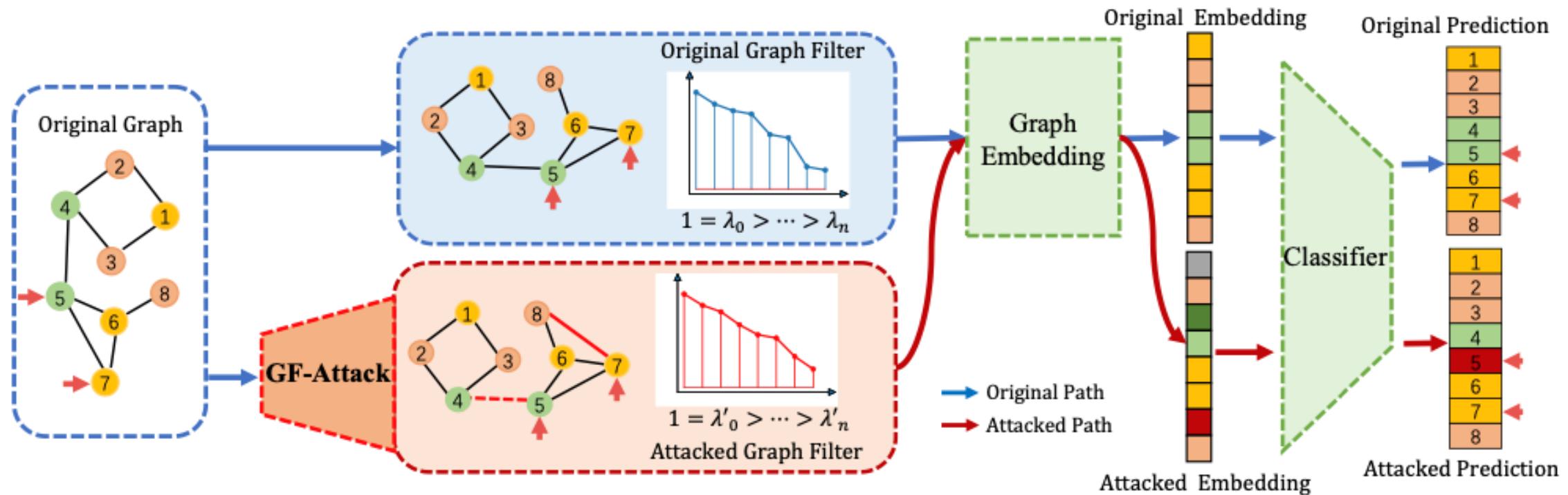
### □ Direct vs. Influence

- Direct: the attacker can directly manipulate the edges/features of the target nodes
- Influence: the attacker can only manipulate other nodes except the targets

### □ Attacker knowledge:

Settings	Parameters	Predictions	Labels	Training Input
<b>White-Box Attack (WBA)</b>	✓	✓	✓	✓
<b>Practical White-box Attack (PWA)</b>		✓	✓	✓
<b>Restrict Black-box Attack (RBA)</b>				✓

# GF-Attack: Restrict Black-box Attack for Graphs



- Understand the various graph embedding model from a new general Graph Signal Processing (GSP) perspective.
- An adversarial framework, *GF-Attack*, is proposed accordingly to perform attacks though graph filters in a RBA fashion.

# GF-Attack: Restrict Black-box Attack for Graphs

□ **Core idea:** constructing the attack damage measuring by attacking the graph filter  $\mathcal{H}$

□ Measuring the quality of output embedding  $Z$  as the  $T$ -rank approximation problem:

$$\mathcal{L}(A', Z) = \|h(S')X - h(S')_T X\|_F^2,$$

$$\begin{aligned} & \arg \max_{A'} \sum_{i=T+1}^n {\lambda'_i}^2 \cdot \sum_{i=T+1}^n \|\mathbf{u}_i^T X\|_2^2, \\ & \text{s.t. } \|A' - A\| = 2\beta. \end{aligned}$$

$\lambda_i$  and  $\mathbf{u}_i$  are an eigen-pair of graph filter  $\mathcal{H}$ .

□ **GCN/SGC as Example:** rewrite the GCN/SGC with  $S = 2I_n - L^{sym}$ :

$$\tilde{X} = (2I_n - L^{sym})^K X, \quad X' = \sigma(\tilde{X}\Theta), \quad L^{sym} = I_n - D^{-\frac{1}{2}}AD^{-\frac{1}{2}} = I_n - \hat{A}$$

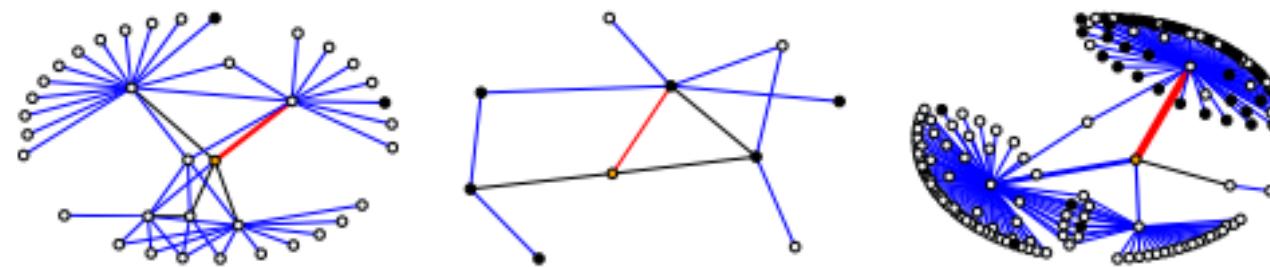
□ The corresponding adversarial attack loss for  $K_{th}$  order GCN/SGC is constructed as:

$$\arg \max_{A'} \sum_{i=T+1}^n (\lambda'_{\hat{A}', i} + 1)^{2K} \cdot \sum_{i=T+1}^n \|\mathbf{u}_{\hat{A}', i}^T X\|_2^2$$

□ A linear time approximation via eigenvalue perturbation theory is used

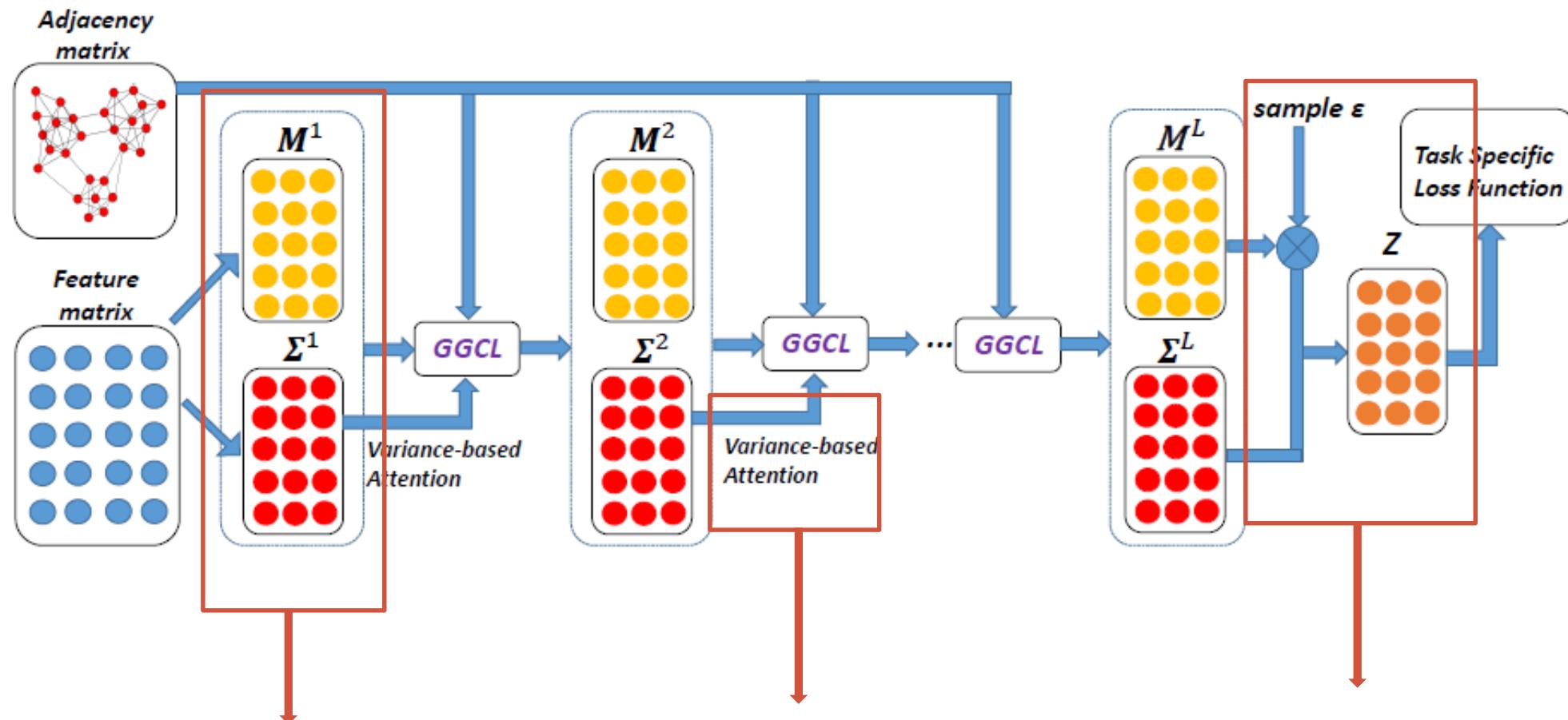
# Robust Graph Convolutional Networks

- How to enhance the robustness of GNNs against adversarial attacks?
- Adversarial attacks in node classification
  - Connect nodes from different communities to confuse the classifier



- Distribution vs. plain vectors
  - Plain vectors cannot adapt to such changes
  - Variances can help to absorb the effects of adversarial changes
  - Gaussian distributions → Hidden representations of nodes

# The Framework of RGCN



Gaussian based representations:  
variance terms absorb the effects of  
adversarial attacks

Attention mechanism:  
Remedy the propagation  
of adversarial attacks

Sampling process: explicitly  
considers mathematical relevance  
between means and variances

# Experimental Results

## □ Node Classification on Clean Datasets

	Cora	Citeseer	Pubmed
GCN	81.5	70.9	79.0
GAT	83.0	72.5	79.0
RGCN	83.1	71.3	79.2

## □ Against Non-targeted Adversarial Attacks

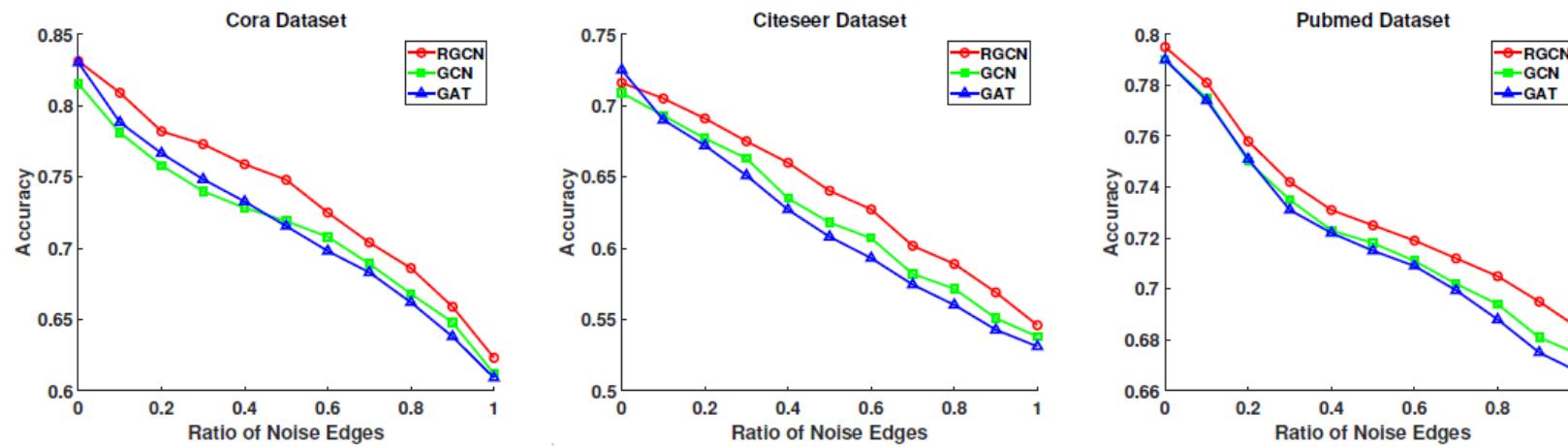
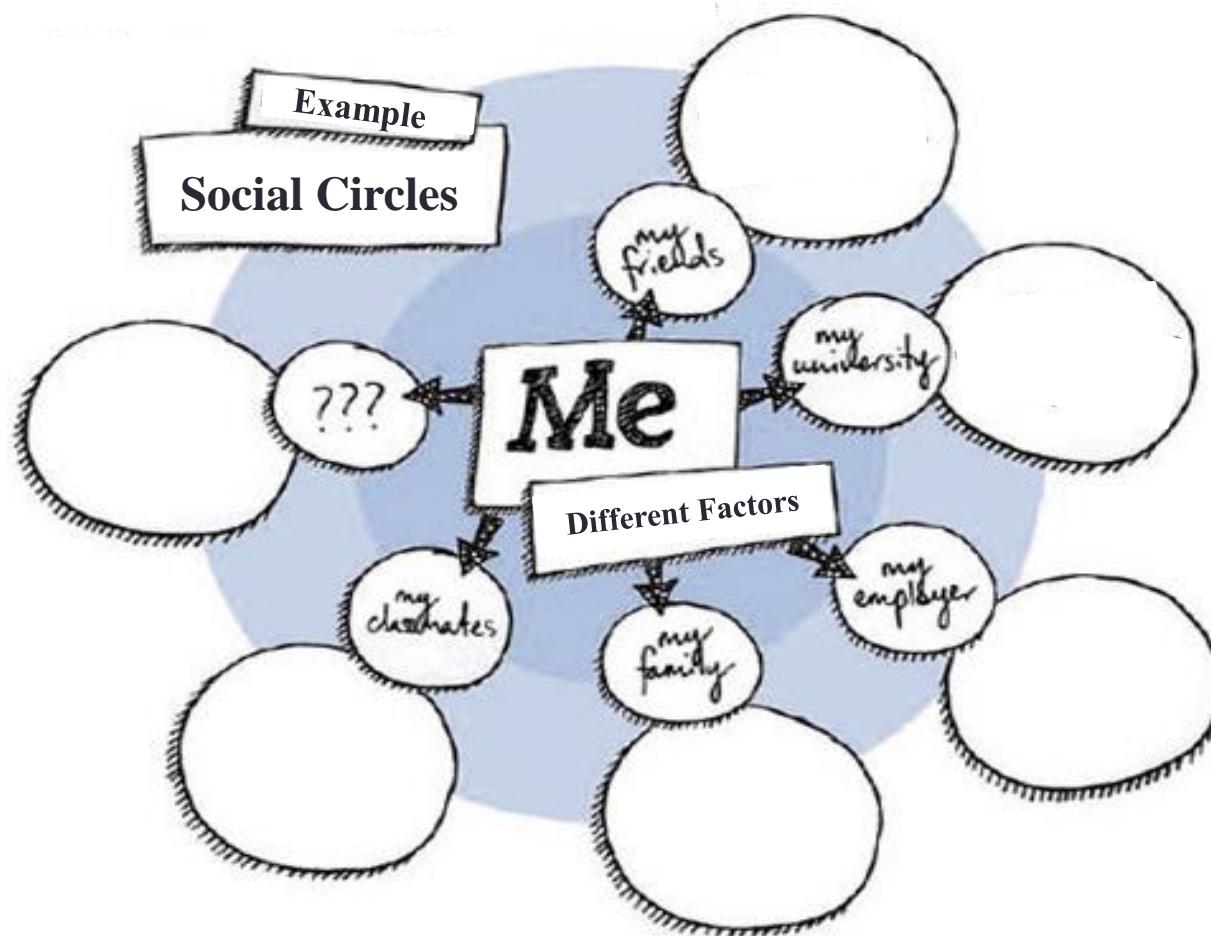


Figure 2: Results of different methods when adopting Random Attack as the attack method.

# Interpretability of GNNs

- A real-world graph is typically formed due to *many* latent factors.

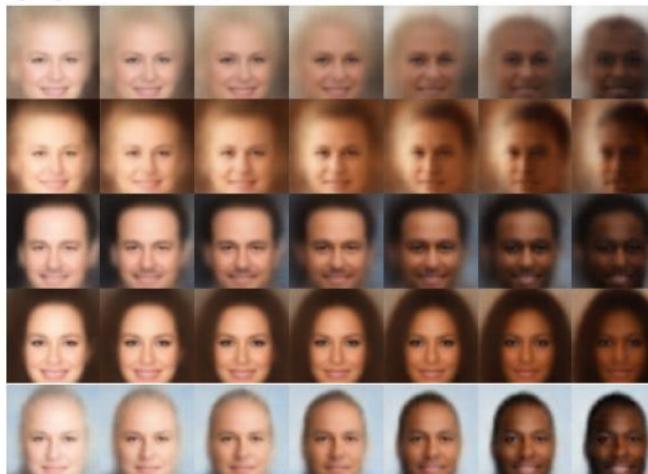


- Existing GNNs/GCNs:
  - A holistic approach, that takes in the *whole* neighborhood to produce a *single* node representation
- We suggest:
  - To disentangle the latent factors  
(By segmenting the heterogeneous parts, and learning multiple factor-specific representations for a node)
  - Robustness (e.g., not overreact to an irrelevant factor) & Interpretability

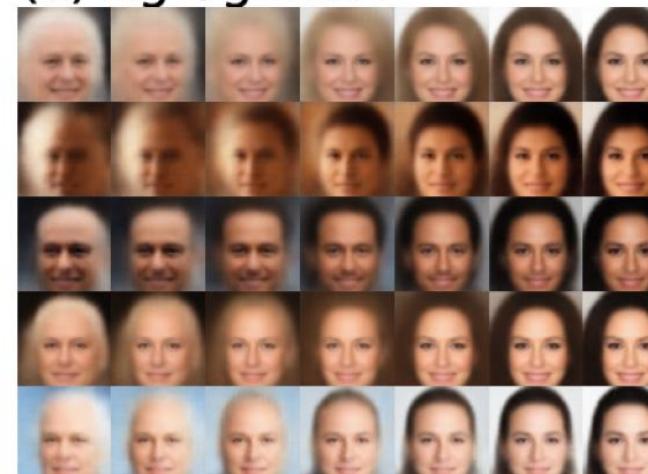
# Disentangled Representation Learning

- That is, we aim to learn disentangled node representations
  - A representation that contains independent components, that describes different aspects (caused by different latent factors) of the observation
- The topic is well studied in the field of computer vision
  - But largely unexplored in the literature of GNNs.

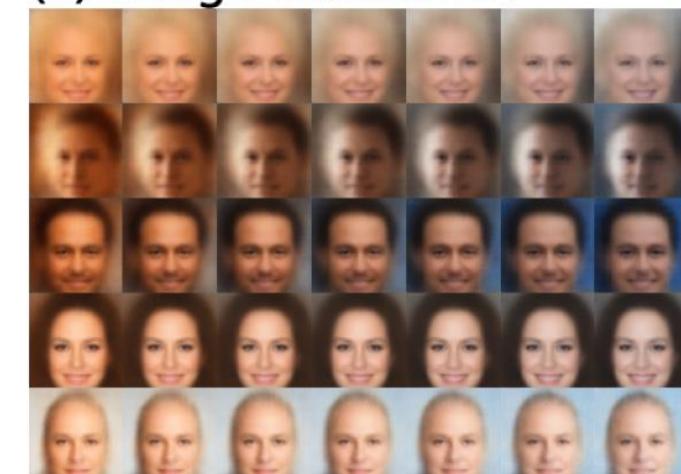
(a) Skin colour



(b) Age/gender



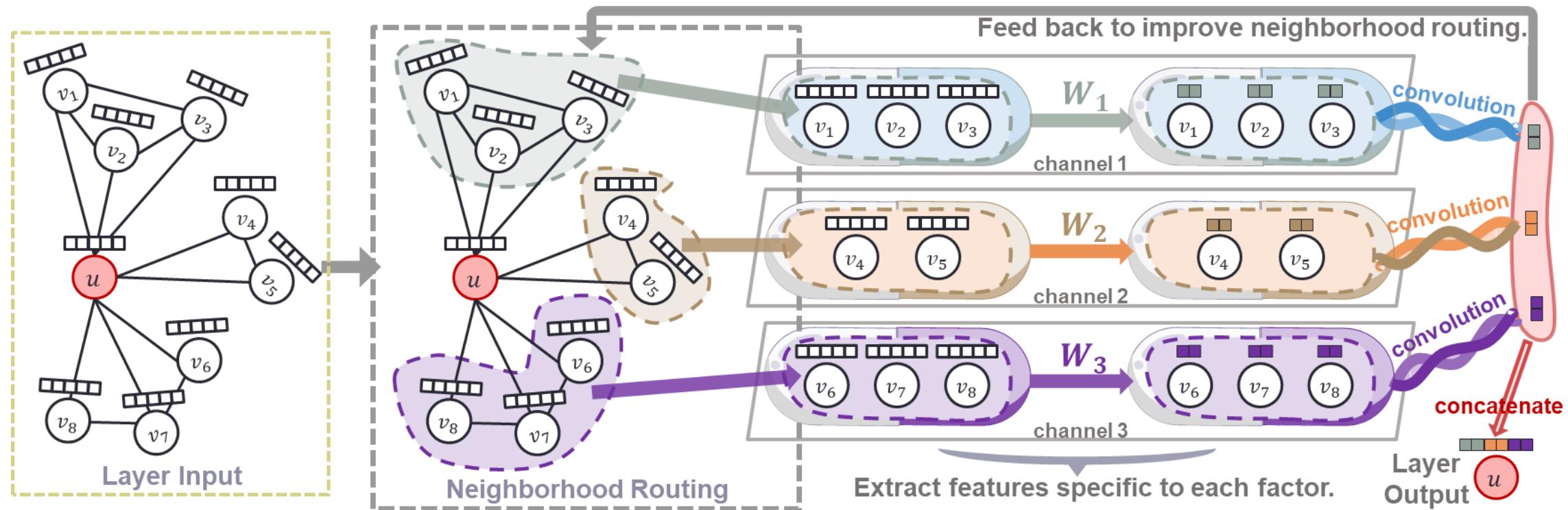
(c) Image saturation



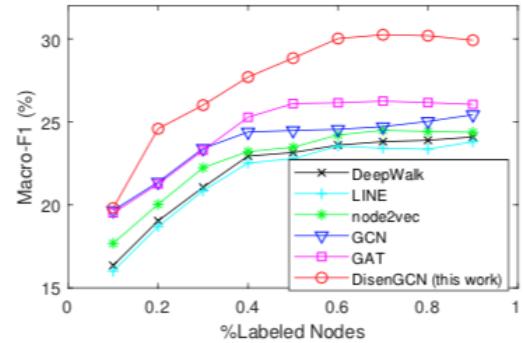
Example: Three dimensions that are related skin color, age/gender, and saturation, respectively.

# Method Overview

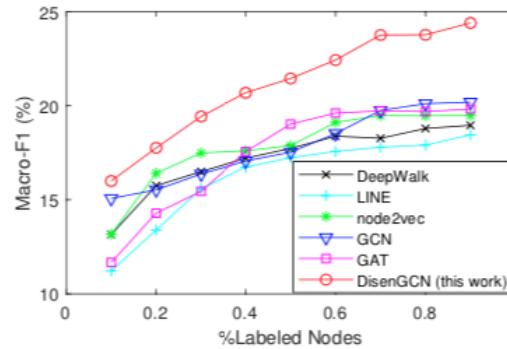
- We present DisenGCN, the *disentangled* graph convolutional network
  - DisenConv, a disentangled multichannel convolutional layer (figure below)
  - Each channel convolves features related with a single latent factor



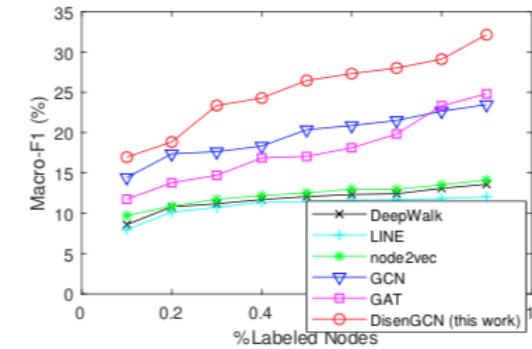
# Results: Multi-label Classification



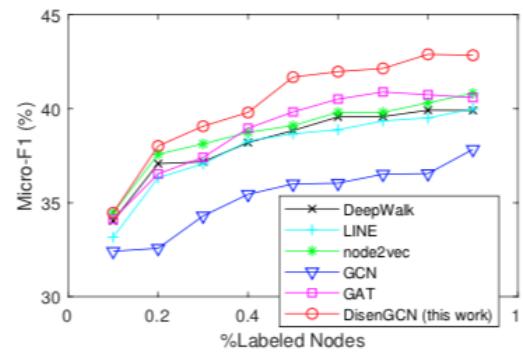
(a) Macro-F1(%), BlogCatalog.



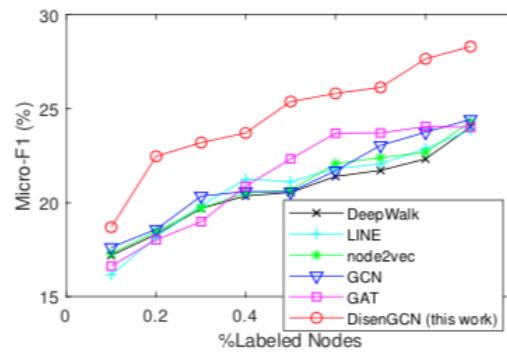
(c) Macro-F1(%), PPI.



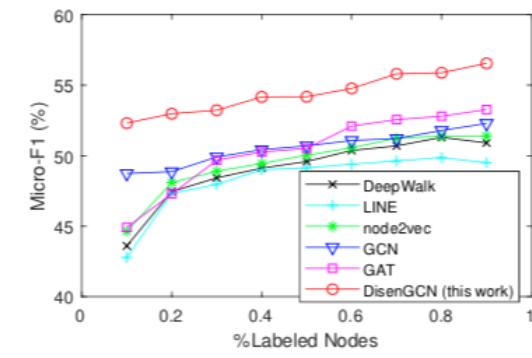
(e) Macro-F1(%), POS.



(b) Micro-F1(%), BlogCatalog.



(d) Micro-F1(%), PPI.



(f) Micro-F1(%), POS.

Figure 2. Macro-F1 and Micro-F1 scores on the multi-label classification tasks. Our approach consistently outperforms the best performing baselines by a large margin, reaching 10% to 20% relative improvement in most cases.

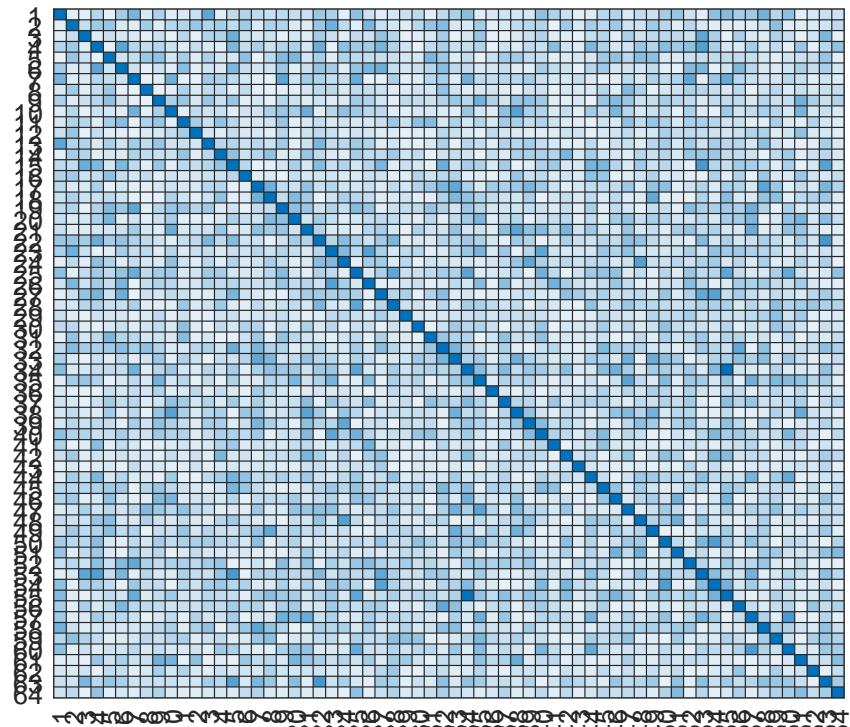
# Results: On Synthetic Graphs

Table 3. Micro-F1 scores on synthetic graphs generated with different numbers of latent factors.

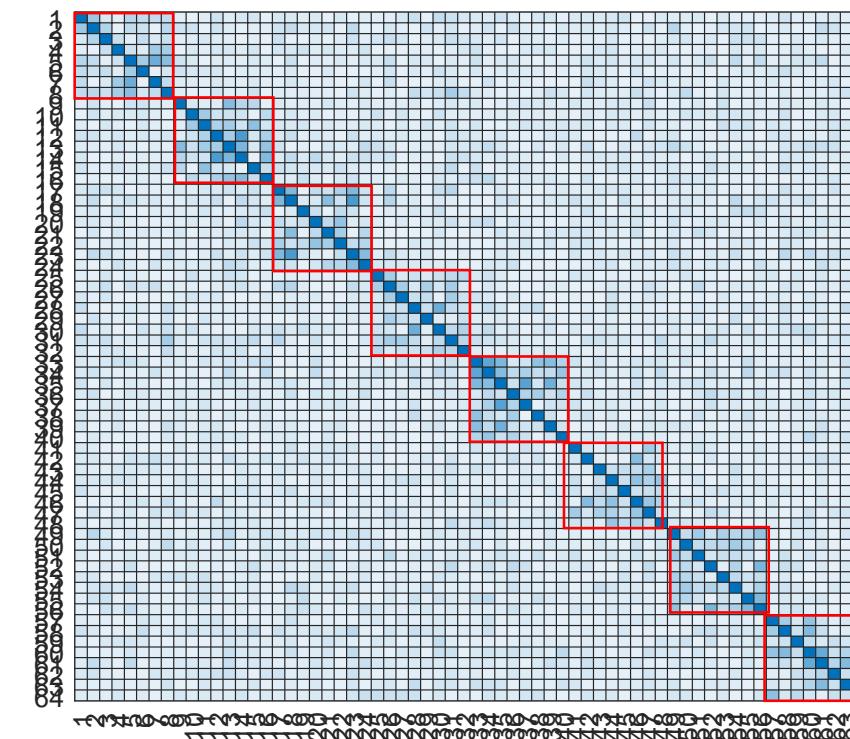
<b>Method</b>	<b>Number of latent factors</b>						
	4	6	8	10	12	14	16
GCN	78.78 ± 1.52	65.73 ± 1.94	46.55 ± 1.55	37.37 ± 1.52	24.49 ± 1.03	18.14 ± 1.50	16.43 ± 0.92
GAT	83.77 ± 2.32	60.89 ± 3.75	45.88 ± 3.79	36.72 ± 3.58	24.77 ± 3.47	20.89 ± 3.57	19.53 ± 3.97
DisenGCN (this work)	<b>93.84</b> ± 1.12	<b>74.68</b> ± 1.92	<b>54.57</b> ± 1.79	<b>43.96</b> ± 1.45	<b>28.17</b> ± 1.22	<b>23.57</b> ± 1.28	<b>21.99</b> ± 1.34
Relative improvement	+12.02%	+13.62%	+17.23%	+17.63%	+13.73%	+12.83%	+12.6%

- Improvement is larger when #factors is relatively large (around 8)
- However, all methods are bad when #factors is extremely large

# Results: Correlations between the Neurons

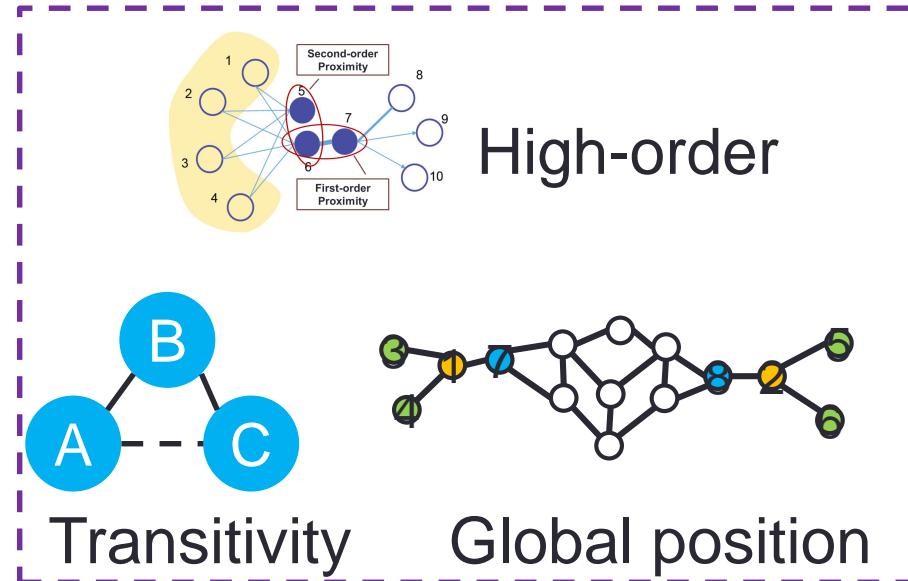


(a) GCN.



(b) DisenGCN (this work).

# Applicability of GNNs/Network embedding



Various network properties

- Link Prediction
- Community Detection
- Node Classification
- Network Distance
- Node Importance
- ...

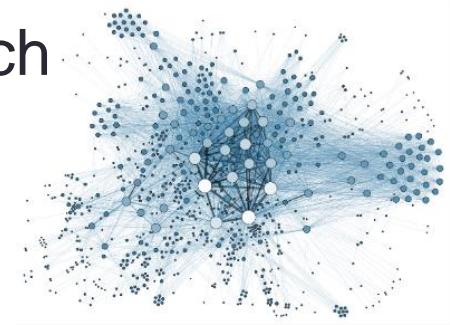
Various applications

- Leading to a large number of hyper-parameters
- Must be carefully tuned

AutoML

# AutoML

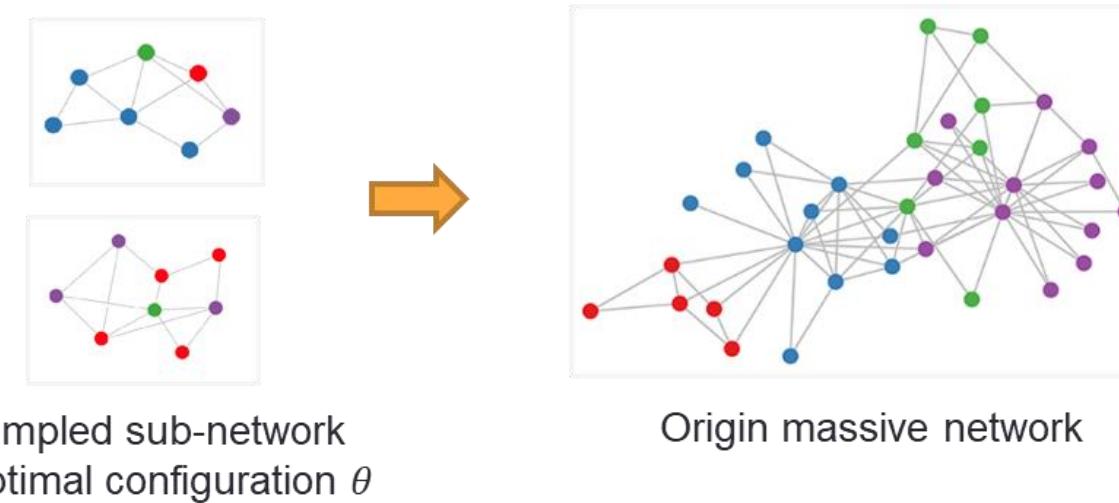
- Ease the adoption of machine learning and reduce the reliance on human experts
  - E.g., hyper-parameter optimization, neural architecture search
- Largely unexplored on **network** data
- Large-scale issue:
  - Complexity of GNNs/Network Embedding is usually at least  $O(E)$ 
    - $E$  is the number of edges (can be 10 billion)
  - Total complexity:  $O(ET)$ ,  $T$  is the times searching for optimal hyperparameters



How to incorporate AutoML into massive GNNs efficiently?

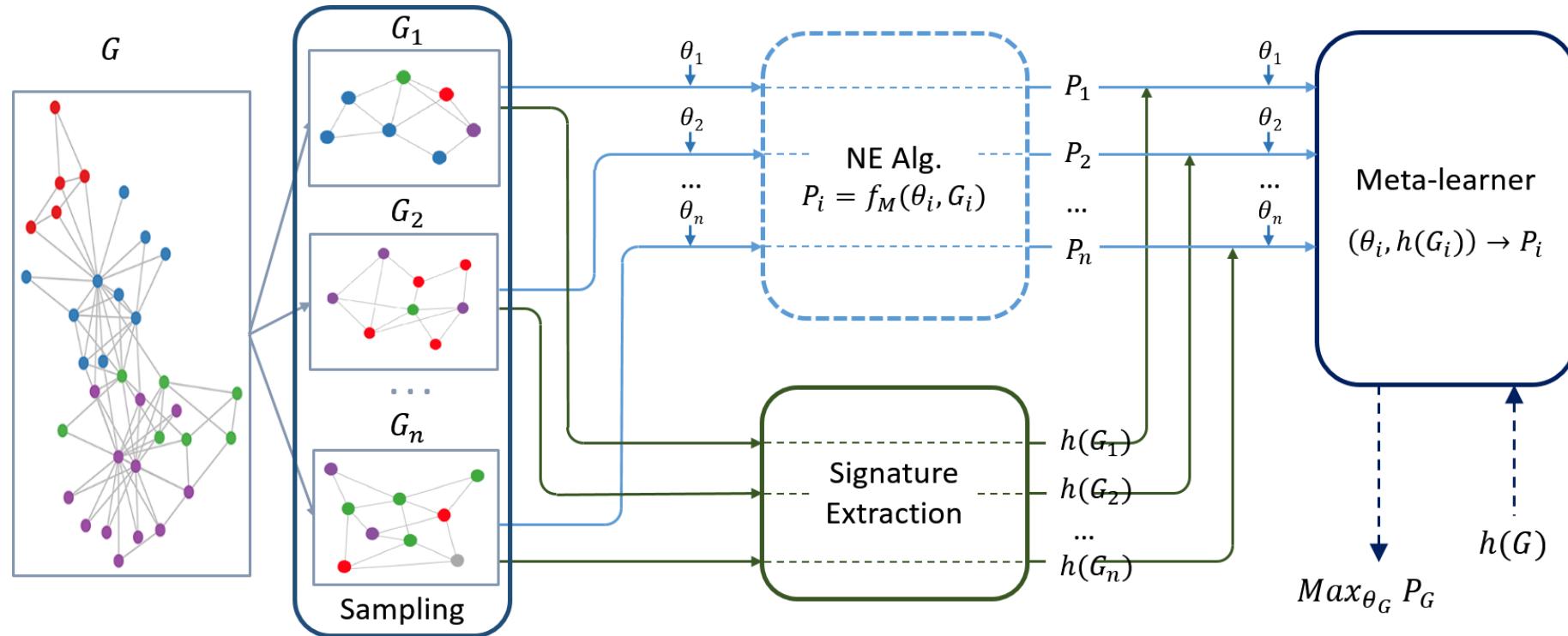
# AutoML for GNN/Network embedding

- A straightforward way: configuration selection on sampled sub-networks



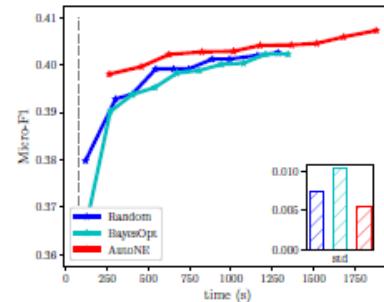
- Transferability
  - $\theta \neq$  optimal configuration on the origin network
- Heterogeneity
  - Several highly heterogeneous components → needs carefully designed sampling

# AutoNE

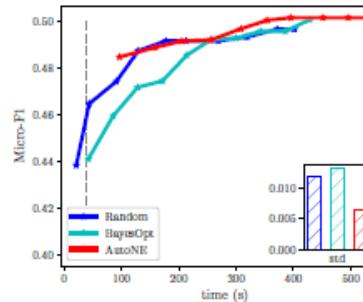


**Transfer the knowledge about optimal hyperparameters from the sub-networks to the original massive network**

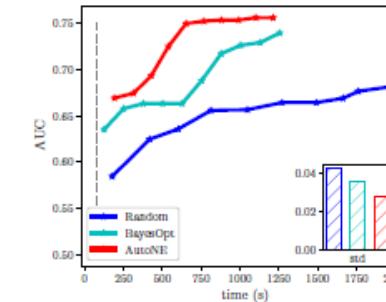
# Experiments: Sampling-Based NE



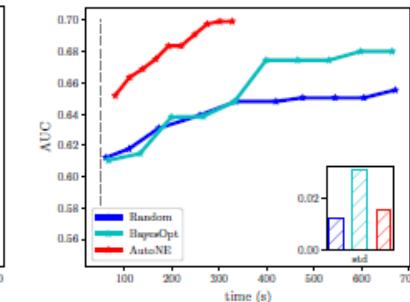
(a) Classification on BlogCatalog



(b) Classification on Wikipedia

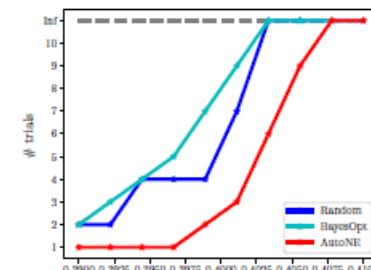


(c) Link prediction on BlogCatalog

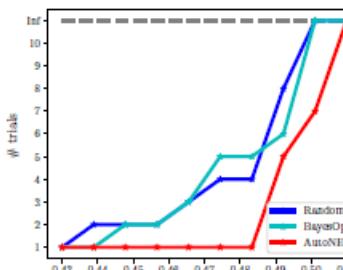


(d) Link prediction on Wikipedia

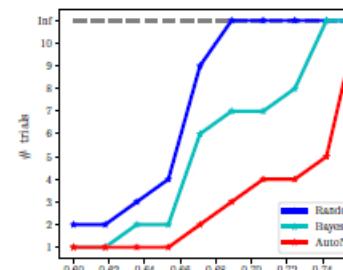
The performance achieved within various time thresholds.



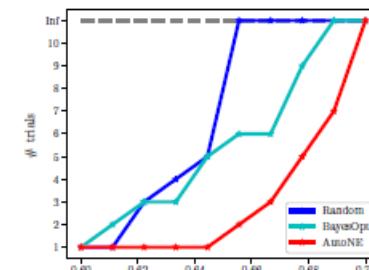
(a) Classification on BlogCatalog



(b) Classification on Wikipedia



(c) Link prediction on BlogCatalog



(d) Link prediction on Wikipedia

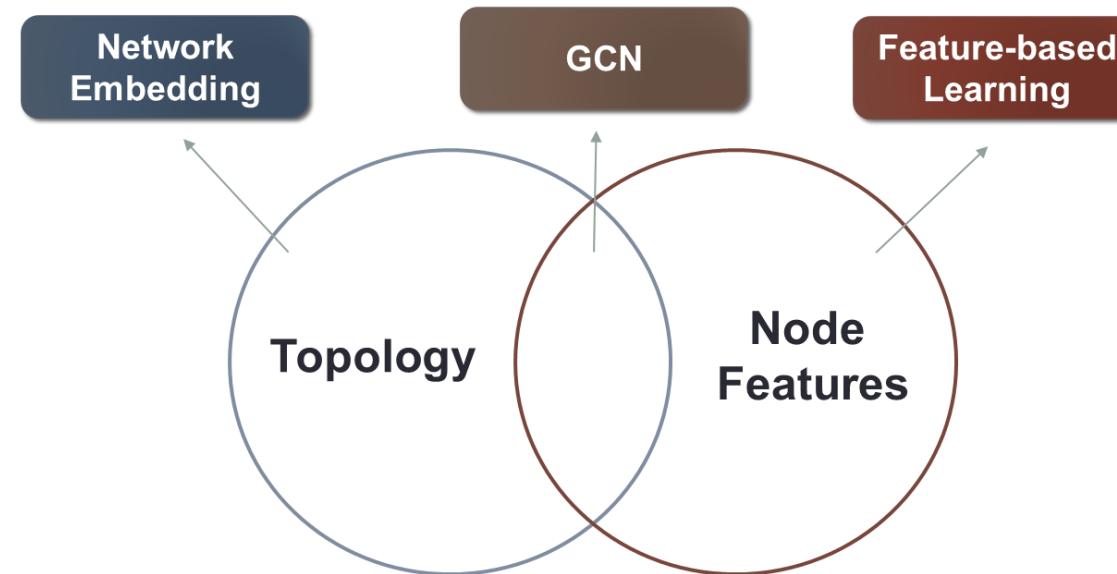
The number of trials to reach a certain performance threshold

# Recap: Graph Neural Networks

- Message-passing framework of GNNs
- Frontiers:
  - Does GNN fuse *feature* and *topology* optimally?
  - Is GNN really a *deep* model?
  - Technical challenges in real applications: robustness, explainability and applicability

# Summaries and Conclusions

- Unsupervised vs. (Semi-)Supervised
- Learning for Networks vs. Learning via Graphs
- Topology-driven vs. Feature-driven
- Both GNN and NE need to treat the counterpart as the baselines



# A Survey on Network Embedding

IEEE TRANSACTIONS ON  
KNOWLEDGE AND  
DATA ENGINEERING

## A Survey on Network Embedding

Issue No. 01 - (preprint vol.)

ISSN: 1041-4347

pp: 1

DOI Bookmark: <http://doi.ieeecomputersociety.org/10.1109/TKDE.2018.2849727>

Peng Cui , Computer Science Department, Tsinghua University, Beijing, Beijing China (e-mail: cuip@tsinghua.edu.cn)

Xiao Wang , Computer Science, Tsinghua University, Beijing, Beijing China (e-mail: wangxiao007@mail.tsinghua.edu.cn)

Jian Pei , School of Computing Science, Simon Fraser University, Burnaby, British Columbia Canada (e-mail: jpei@cs.sfu.ca)

Wenwu Zhu , Department of Computer Science, Tsinghua University, Beijing, Beijing China (e-mail: wwzhu@tsinghua.edu.cn)

### ABSTRACT

Network embedding assigns nodes in a network to low-dimensional representations and effectively preserves the network structure. Recently, a significant amount of progresses have been made toward this emerging network analysis paradigm. In this survey, we focus on categorizing and then reviewing the current development on network embedding methods, and point out its future research directions. We first summarize the motivation of network embedding. We discuss the classical graph embedding algorithms and their relationship with network embedding. Afterwards and primarily, we provide a comprehensive overview of a large number of network embedding methods in a systematic manner, covering the structure- and property-preserving network embedding methods, the network embedding methods with side information and the advanced information preserving network embedding methods. Moreover, several evaluation approaches for network embedding and some useful online resources, including the network data sets and softwares, are reviewed, too. Finally, we discuss the framework of exploiting these network embedding methods to build an effective system and point out some potential future directions.

Peng Cui, Xiao Wang, Jian Pei, Wenwu Zhu. A Survey on Network Embedding. *IEEE TKDE*, 2018.

# Deep Learning on Graphs: A Survey

Journals & Magazines > IEEE Transactions on Knowledg... > Early Access ?

## Deep Learning on Graphs: A Survey

Publisher: IEEE

Cite This

PDF

Ziwei Zhang ; Peng Cui ; Wenwu Zhu All Authors

3  
Paper Citations

1508  
Full Text Views



### Abstract

#### Abstract:

Deep learning has been shown to be successful in a number of domains, ranging from acoustics, images, to natural language processing. However, applying deep learning to the ubiquitous graph data is non-trivial because of the unique characteristics of graphs. Recently, substantial research efforts have been devoted to applying deep learning methods to graphs, resulting in beneficial advances in graph analysis techniques. In this survey, we comprehensively review the different types of deep learning methods on graphs. We divide the existing methods into five categories based on their model architectures and training strategies: graph recurrent neural networks, graph convolutional networks, graph autoencoders, graph reinforcement learning, and graph adversarial methods. We then provide a comprehensive overview of these methods in a systematic manner mainly by following their development history. We also analyze the differences and compositions of different methods. Finally, we briefly outline the applications in which they have been used and discuss potential future research directions.

Published in: IEEE Transactions on Knowledge and Data Engineering (Early Access)

Ziwei Zhang, Peng Cui, Wenwu Zhu. Deep Learning on Graphs: A Survey. *IEEE TKDE*, 2020.

# Thanks!

Peng Cui

[cui@tsinghua.edu.cn](mailto:cui@tsinghua.edu.cn)

<http://pengcui.thumedialab.com>

---



*media and network lab*